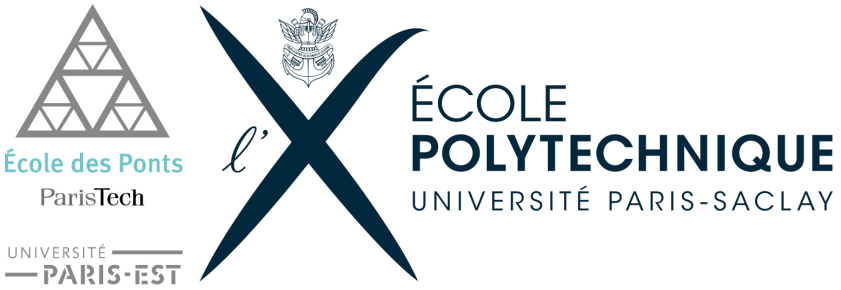

Laboratoire Navier, ENPC, IFSTTAR, CNRS

Rapport de stage de recherche 3A

Implémentation d'un schéma d'homogénéisation en milieu de gradient supérieur

---

Auteur :
Jun Zeng

Tuteur :
Arthur Lebée

27 août 2017

Table des matières

Table des figures

Liste des tableaux

Remerciement

Je tiens à remercier tout particulièrement et à témoigner ma reconnaissance à Arthur Lebée pour l'aide qu'il m'a apportée durant ces quatre mois. Sa disponibilité, sa pédagogie et ses conseils m'ont été précieux dans le théorie de l'homogénéisation et l'implémentation du code de Fortran 77 et Python dans Abaqus. Nos discussions régulières assure le progrès sans répit du projet de recherche scientifique dans ce stage.

Je remercie par ailleurs l'auteur de les cours principaux de référence : Karam Sab. Les informations sont précieuses pour comprendre le principe d'homogénéisation dans un milieu périodique, me permis de poursuivre ce projet dans les meilleures conditions.

1 Introduction

Un milieu périodique est un matériau occupant entièrement l'espace tel qu'il existe une cellule de base qui reproduit ce milieu par des translations géométrique. Alors on peut dire que la cellule est la plus petite unité de la périodicité.

Pour identifier la loi de comportement macroscopique de ce milieu périodique, nous faisons une homogénéisation dedans. Suite du cours de l'homogénéisation de Karam Sab [2], l'hypothèse de l'homogénéisation "Classique" est de supposer qu'il existe une relation linéaire entre la déformation macroscopique (E_{ij}) et la contrainte moyenne et introduisons aussi un champ de tenseur d'élasticité macroscopique qu'on appelle (C^{hom}_{ijkl}). Alors à l'aide de cette relation linéaire, il serait commode de remplacer ce matériau périodique par un matériau élastique "homogène équivalent" caractérisé par le tenseur d'élasticité mentionné qu'on l'appelle le tenseur de Cauchy. En effet, si on vise à faire un calcul des éléments finis en prenant la taille des éléments qui soit plus petite que la taille de la cellule, le calcul est très coûteux. Dans la pratique, nous choisissons l'élément dont la taille soit beaucoup plus grande que la cellule et beaucoup plus petite que la taille du milieu et cette homogénéisation nous permet d'assurer l'existence de ce type d'élément.

Nous trouvons que l'aide de l'homogénéisation "Classique", les calcul de l'échelle macroscopique et microscopique peuvent être séparés. Mais dans le cas où le tenseur (C^{hom}_{ijkl}) est singulier, c'est-à-dire certains coefficients dans le tenseur sont nuls et ce tenseur n'est plus inversible, nous ne pouvons pas définir le tenseur de souplesse (S^{hom}_{ijkl}) associé et il reste moins de crédibilité pour ce genre d'homogénéisation.

Nous commençons à contester la relation linéaire entre la contrainte moyenne et la déformation macroscopique et nous introduisons le schéma d'homogénéisation en gradient d'ordre supérieur, décrit par V. P. Smyshlyaev et K. D. Cherednichenko [3]. Ce nouveau schéma d'homogénéisation est afin de régler le problème dans le cas où (C^{hom}_{ijkl}) n'est plus inversible. Le principe de cette homogénéisation est de continuer développement asymptotique utilisé pour homogénéisation "Classique". Alors l'homogénéisation en gradient d'ordre supérieur implique que le deuxième terme du déplacement dans le développement limité soit linéaire au gradient des déformations $K_{ijk} = \partial_{Y_k} E_{ij}$, c'est-à-dire le second gradient du déplacement. Dans la suite du rapport, on l'appelle directement second gradient.

En effet, il n'y a que le travail théorique maintenant pour le schéma d'homogénéisation en gradient d'ordre supérieur. C'est intéressant d'implémenter cette théorie afin de traiter les résultats avec Abaqus. Une validation avec un modèle simple est utile pour vérifier notre calcul.

Dans ce rapport, nous présentons premièrement le schéma d'homogénéisation en gradient d'ordre supérieur pour le cas du second gradient. Ensuite nous faisons une implémentation de cette théorie en traitant les données calculées par Abaqus dans un modèle de déformation plane. Pour vérifier notre algorithme, nous proposons aussi une validation avec un modèle stratifié simple qui peut être calculé analytiquement. Finalement, nous faisons une application en utilisant cette homogénéisation dans un modèle de pantographe, et expliquons l'effet de second gradient et les propriétés intéressantes de la structure de pantographe.

Ce stage a été financé par le projet PEPS "Matériaux à gradients de déformation, matériaux à gradients de contraintes", collaboration entre Pierre Seppecher (Université de Toulon, IMATH), Arthur Lebée et Karam Sab (Laboratoire Navier, Ecole des Ponts ParisTech, IFSTTAR, CNRS) financée par le CNRS. Les structures pantographiques ont été suggérées par Pierre Seppecher.

2 Schéma d'homogénéisation en gradient d'ordre supérieur

2.1 Principe

Le schéma d'homogénéisation en gradient d'ordre supérieur consiste à exprimer le déplacement en développement asymptotique à l'échelle intermédiaire. Dans ce processus, nous supposons que le champ du déplacement soit en fonction non seulement d'une variable macroscopique $Y_i = x_i/T$, mais aussi d'une variable microscopique $y_i = x_i/t$, dans laquelle T est l'échelle macroscopique qu'on observe le phénomène macroscopique (la taille du milieu) et t est la longueur caractéristique de la cellule. Nous proposons aussi un paramètre $\epsilon = t/T$ et ce paramètre nous permet d'écrire l'opérateur de la dérivée partielle :

$$\nabla_x = \frac{1}{T}(\nabla_Y + \frac{1}{\epsilon}\nabla_y) \tag{1}$$

le déplacement est écrit en développement asymptotique :

$$u_i(Y_p, y_q) = u_i^0(Y_p, y_q) + \epsilon u_i^1(Y_p, y_q) + \epsilon^2 u_i^2(Y_p, y_q) + ...$$

En substituant le champ du déplacement par ce développement asymptotique dans les équations du problème d'élasticité, nous pouvons obtenir une série des équations des différents ordres, Les équations de l'ordre plus petit nous implique que u_i^0 est une fonction macroscopique et on note $u_i^0 = U_i^0(Y_p)$ dans les calculs suivants.

2.2 Problème du premier ordre

Premièrement, nous nous concentrons sur le problème auxiliaire du premier ordre, c'est-à-dire dans ce cas, nous considérons que le déplacement inconnu $u_i(Y_p, y_q)$ est composée des deux termes

$$u_i(Y_p, y_q) = U_i^0(Y_p) + \epsilon u_i^1(Y_p, y_q) \tag{2}$$

et les équations du problème s'écrivent comme :

$$\begin{cases} \partial_{y_j}\sigma_{ij}^0 = 0 \\ \sigma_{ij}^0 = C_{ijkl}(y_p)\varepsilon_{kl}^0 \\ \varepsilon_{ij}^0 = E_{ij}^0 + \frac{1}{2}\left(\partial_{y_j}u_i^1 + \partial_{y_i}u_j^1\right) \\ u^1 \text{ est une fonction périodique de } y_i \end{cases} \tag{3}$$

Le champs $U_i^0(Y_p)$ est constant par rapport à y_i dans la cellule et nous notons E_{ij}^0 pour la déformation macroscopique :

$$E_{ij}^0 = \frac{1}{2}\left(\partial_{Y_j}U_i^0 + \partial_{Y_i}U_j^0\right)$$

alors la solution complète du problème (3) est écrite sous forme d'une superposition linéaire :

$$u_i^1(Y_p, y_q) = u_{ikl}^{\nabla}(y_q)E_{lk}^0(Y_p) + U_i^1(Y_p) \tag{4}$$

la constante $U_i^1(Y_p)$ ici présente la déformation moyenne dans chaque cellule et on impose $\langle u_{ikl}^{\nabla}\rangle = 0$. Dans les équations (3), nous voyons la précontrainte $C_{ijkl}(y_p)E_{ij}^0$ et les conditions aux limites sont indéterminées. Pour mieux identifier ces conditions aux limites et éviter de calculer la précontrainte, nous intégrons l'équation $\varepsilon_{ij}^0 = E_{ij}^0 + \frac{1}{2}\left(\partial_{y_j}u_i^1 + \partial_{y_i}u_j^1\right)$, alors on obtient :

$$\xi_i^0 = E_{ij}^0 y_j + u_i^1 \tag{5}$$

Or $E_{ij}y_j$ est périodique entre les cellules, alors le problème équivalent du premier ordre s'écrit avec une nouvelle périodicité de ξ_i :

$$\begin{cases} \partial_{y_j}\sigma^0_{ij} = 0 \\ \sigma^0_{ij} = C_{ijkl}(y_p)\varepsilon^0_{kl} \\ \varepsilon^\nabla_{kl} = \nabla^s\xi \\ \xi^+_i(y_i = l_i) - \xi^-_i(y_i = l_i) = E_{ij}l_i \end{cases} \tag{6}$$

et l'inconnu de ce nouvelle forme du problème devient ξ^0_i. l_i présente la longueur de la cellule dans la direction i. $\xi^+_i(y_i = l_i)$ et $\xi^-_i(y_i = l_i)$ présente les dans l'interface entre les cellules. les signes $+$ et $-$ signifient la direction du vecteur normal du plan.

Homogénéisation de Cauchy

Nous voulons établir une relation entre la déformation macroscopique et la contrainte moyenne. D'abord on définit :

$$\varepsilon^0_{ij} = \left(I_{ijkl} + \frac{1}{2}\left(\partial_{y_j}u^\nabla_{ikl} + \partial_{y_i}u^\nabla_{jkl}\right)\right) E_{kl} = A^\nabla_{ijkl}(y_p)E_{kl} \tag{7}$$

où I_{ijkl} est un tenseur d'ordre 4 :

$$I_{ijkl} = \frac{1}{2}(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$$

Le tenseur de localisation $A^\nabla_{ijkl}(y_p)$ a les symétries mineures suivantes :

$$A^\nabla_{ijkl}(y_p) = A^\nabla_{jikl}(y_p) = A^\nabla_{ijlk}(y_p)$$

mais il ne vérifie pas en général la symétrie majeure :

$$A^\nabla_{ijkl}(y_p) \neq A^\nabla_{klij}(y_p)$$

A l'aide de l'équation (7), on établit la relation linéaire entre la contrainte moyenne et la déformation macroscopique :

$$\Sigma^0_{ij} = \langle\sigma^0_{ij}\rangle = \langle C_{ijmn}(y_p)A^\nabla_{mnkl}(y_p)\rangle E_{kl} = C^{hom}_{ijkl}E_{kl} \tag{8}$$

Donc :

$$C^{hom}_{ijkl} = \langle C_{ijmn}(y_p)A^\nabla_{mnkl}(y_p)\rangle \tag{9}$$

et

$$(S^{hom}_{ijkl}) = (C^{hom}_{ijkl})^{-1} \tag{10}$$

Nous notons aussi le tenseur de localisation des contraintes B^∇_{ijkl} dans le problème du premier ordre, reliant les contraintes microscopiques σ^0_{ij} à la déformation macroscopique E^0_{ij} :

$$\sigma^0_{ij} = B^\nabla_{ijkl}E^0_{kl}$$

dans laquelle le tenseur B_{ijkl} s'écrit par :

$$B^\nabla_{ijkl} = C_{ijmn}A^\nabla_{mnkl} \tag{11}$$

Homogénéisation de 2ème gradient

Dans le calcul suivant, nous écrivons l'énergie moyenne de la déformation en développement asymptotique en fonction de la déformation macroscopique et gradient des déformations (second gradient). En prenant la troncature de déplacement :

$$u_i(Y_p, y_q) = u_i^0(Y_p, y_q) + \epsilon u_i^1(Y_p, y_q) = U_i(Y_p) + \epsilon u_{ikl}^{\nabla}(y_q) E_{lk}(Y_p)$$

alors la déformation totale peut être calculée par la dérivé partielle de l'operateur (1), alors on obtient

$$T\varepsilon_x(u_i) = \varepsilon_Y(u_i) + \frac{1}{\epsilon}\varepsilon_y(u_i)$$

$$T\varepsilon_x(u_i) = A_{ijkl}^{\nabla} E_{kl} + \epsilon \frac{1}{2}(u_{ikl}^{\nabla}\partial_{Y_j} E_{lk} + u_{jkl}^{\nabla}\partial_{Y_i} E_{lk}) \tag{12}$$

alors nous écrivons avec une forme factorisée en utilisant la symbole de Kronecker à partir de l'équation (12) :

$$T\varepsilon_x(u_i) = A_{ijkl}^{\nabla} E_{kl} + \epsilon \frac{1}{2}(u_{ikl}^{\nabla}\delta_{jm} + u_{jkl}^{\nabla}\delta_{im})\partial_{Y_m} E_{lk} \tag{13}$$

A partir de l'équation (13), c'est possible pour nous de calculer l'énergie de la déformation dans la cellule :

$$\frac{1}{2T^2}\langle \partial_{x_i} u_j C_{ijkl} \partial_{x_k} u_l \rangle = \frac{1}{2T^2}(E_{ij} C_{ijkl}^{hom} E_{kl} + \epsilon^2 \partial_{Y_k} E_{ij} D_{ijklmn} \partial_{Y_l} E_{mn}) \tag{14}$$

où

$$D_{ijklmn} = \langle u_{qij}^{\nabla}(y_p) C_{qklr}(y_p) u_{rmn}^{\nabla}(y_p) \rangle \tag{15}$$

D'une vue macroscopique, l'énergie moyenne de la cellule contient une terme en fonction de la gradient des déformations. Le tenseur correspondant est positif et semi-defini. Notons que D est nulle si le matériau est homogène et cela nous fournit une manière à faire une validation sur l'implémentation.

2.3 Problème du deuxième ordre

Pour le calcul du deuxième ordre, alors les équations du problème sont écrites :

$$\begin{cases} \partial_{y_j}\sigma_{ij}^1 + \partial_{Y_j}\sigma_{ij}^0 + F_i = 0 \\ \sigma_{ij}^1 = C_{ijkl}(y_p)\varepsilon_{kl}^1 \\ \varepsilon_{ij}^1 = \frac{1}{2}\left(\partial_{Y_j} u_i^1 + \partial_{Y_i} u_j^1\right) + \frac{1}{2}\left(\partial_{y_j} u_i^2 + \partial_{y_i} u_j^2\right) \\ u^2 \text{ est une fonction périodique de } y_i \end{cases}$$

Nous faisons une intégrale dans l'espace d'une cellule pour la première équation, alors on obtient :

$$\frac{1}{s_{cell}}\int \partial_{y_j}\sigma_{ij}^1 + \frac{1}{s_{cell}}\int \partial_{Y_j}\sigma_{ij}^0 + \frac{1}{s_{cell}}\int F_i = 0 \tag{16}$$

dans laquelle S_{cell} présente l'air de la cellule. Grâce au théorème de divergence appliqué à σ_{ij}^1, on obtient $\int_{s_{cell}} \partial_{y_j}\sigma_{ij}^1 = 0$, or $\int_{s_{cell}} \partial_{Y_j}\sigma_{ij}^0 = \partial_{Y_j}\int_{s_{cell}} \sigma_{ij}^0$, alors nous obtenons une force volumique :

$$F_i = -\partial_{Y_j}\Sigma_{ij}^0$$

Donc on a une nouvelle forme du problème de deuxième ordre :

$$\begin{cases} \partial_{y_j}\sigma^1_{ij} + \partial_{Y_j}(\sigma^0_{ij} - \langle\sigma^0_{ij}\rangle) = 0 \\ \sigma^1_{ij} = C_{ijkl}(y_p)\varepsilon^1_{kl} \\ \varepsilon^1_{ij} = \frac{1}{2}\left(\partial_{Y_j}u^1_i + \partial_{Y_i}u^1_j\right) + \frac{1}{2}\left(\partial_{y_j}u^2_i + \partial_{y_i}u^2_j\right) \\ u^2 \text{ est une fonction périodique de } y_i \end{cases}$$

En considérant la dérivée partielle par rapport à Y_k, nous introduisons le gradient des déformations macroscopique K_{ijk} qu'on commencera à voir dans le calcul du tenseur (D_{ijklmn}) :

$$K_{ijk}(Y_p) = \partial_{Y_k}E_{ij}$$

alors avec cette notation, les équations du problème de l'ordre 2 sont écrites :

$$\begin{cases} \partial_{y_j}\sigma^1_{ij} + (C_{ijmn}A^{\nabla}_{mnkl} - C^{hom}_{ijkl})K_{lkj} = 0 \\ \sigma^1_{ij} = C_{ijkl}(y_p)\varepsilon^1_{kl} \\ \varepsilon^1_{ij} = \frac{1}{2}(u^{\nabla}_{ikl}(y_q)\delta_{jm} + u^{\nabla}_{jkl}\delta_{im})K_{lkm} + \frac{1}{2}\left(\partial_{y_j}u^2_i + \partial_{y_i}u^2_j\right) \\ u^2 \text{ est une fonction périodique de } y_i \end{cases} \tag{17}$$

A partir des équations au-dessus, nous voyons que d'un point de vue numérique, la cellule est soumise à une précontrainte dans les différents nœuds et des forces volumiques aux différents points d'intégration. La précontraintes et les forces volumiques seront calculées en fonction de (u^{∇}_{ijk}) qui dépend des résultats du calcul du premier ordre.

Dans ce cas, la solution complète s'écrit dans une forme de superposition :

$$u^2_i(Y_p, y_q) = u^{\nabla^2}_{imlk}K_{klm}(Y_p)$$

dans laquelle Y_p et y_q représentent les coordonnées macroscopique et microscopique. De plus, nous choisissons une normalisation pour u^{∇^2} :

$$\langle u^{\nabla^2}_{imlk}\rangle = 0$$

Il nous permet d'introduire une déformation associée avec $u^2_i(Y_p, y_q)$:

$$\varepsilon^1_{ij} = \frac{1}{2}((u^{\nabla}_{ikl}\delta_{jm} + u^{\nabla}_{jkl}\delta_{im}) + (\partial_{y_j}u^{\nabla^2}_{imlk}(y_q) + \partial_{y_i}u^{\nabla^2}_{jmlk}(y_q)))K_{lkm} = A^{\nabla^2}_{ijmlk}K_{lkm}$$

dans laquelle nous définissons le tenseur de localisation de déformations dans le calcul du deuxième ordre :

$$A^{\nabla^2}_{ijmlk}(y_q) = \frac{1}{2}((u^{\nabla}_{ikl}\delta_{jm} + u^{\nabla}_{jkl}\delta_{im}) + (\partial_{y_j}u^{\nabla^2}_{imlk}(y_q) + \partial_{y_i}u^{\nabla^2}_{jmlk}(y_q)))$$

Nous notons aussi le tenseur de localisation des contraintes $B^{\nabla^2}_{ijklm}$ dans le problème du deuxième ordre :

$$B^{\nabla^2}_{ijklm} = C_{ijpq}A^{\nabla^2}_{pqklm} \tag{18}$$

Homogénéisation de 3ème gradient

Premièrement, la fonction du déplacement est écrite avec un développement asymptotique, en considérant le terme u^2_i :

$$u_i = U_i + \epsilon u^1_i + \epsilon^2 u^2_i = U_i(Y_p) + \epsilon u^{\nabla}_{ijk}(y_q)E_{lk}(Y_p) + \epsilon^2 u^{\nabla^2}_{imlk}(y_q)\partial_{Y_m}E_{kl}(Y_p)$$

alors c'est possible de calculer la déformation totale en utilisant l'opérateur de la dérivée macroscopique et microscopique (1) :

$$T\varepsilon_x(u_i) = (\nabla_Y + \frac{1}{\epsilon}\nabla_y)(u_i) \tag{19}$$

or

$$\begin{aligned}
&(\nabla_Y + \frac{1}{\epsilon}\nabla_y)(u_i) \\
=&(\nabla_Y + \frac{1}{\epsilon}\nabla_y)(U_i(Y_p) + \epsilon u^{\nabla}_{ijk}(y_q)E_{lk}(Y_p) + \epsilon^2 u^{\nabla^2}_{imlk}(y_q)\partial_{Y_m}E_{kl}(Y_p)) \\
=&\Big(\nabla_Y U_i(Y_p) + \epsilon u^{\nabla}_{ijk}\nabla_Y E_{lk}(Y_p) + \epsilon^2 u^{\nabla^2}_{imk}(y_q)\nabla_Y K_{lkm}\Big) + \\
&\Big(\nabla_y u^{\nabla}_{ijk}E_{kl}(Y_p) + \epsilon\nabla_y u^{\nabla^2}_{imk}(y_q)K_{lkm}\Big) \\
=&A^{\nabla}_{ijkl}E_{kl} + \epsilon A^{\nabla^2}_{ijklm}E_{ml,k} + \epsilon^2\frac{1}{2}(u^{\nabla^2}_{iklm}\delta_{in} + u^{\nabla^2}_{jklm}\delta_{in})\partial_{Y_n}K_{mlk}
\end{aligned} \tag{20}$$

car on a ;

$$\begin{aligned}
A^{\nabla}_{ijkl}E_{kl} &= \nabla_Y u_i(Y_p) + \nabla_y u^{\nabla}_{ijk}E_{kl}(Y_p) \\
\epsilon A^{\nabla^2}_{ijklm}K_{mlk} &= \epsilon u^{\nabla}_{ijk}\nabla_Y E_{lk}(Y_p) + \epsilon\nabla_y u^{\nabla^2}_{imk}(y_q)K_{lkm} \\
\epsilon^2\frac{1}{2}(u^{\nabla^2}_{iklm}\delta_{in} + u^{\nabla^2}_{jklm}\delta_{in})\partial_{Y_n}K_{mlk} &= \epsilon^2 u^{\nabla^2}_{imk}(y_q)\nabla_Y K_{lkm}
\end{aligned}$$

L'énergie de la déformation est possible d'être calculée à partir de cette relation à l'aide des équations (19) et (20) :

$$\begin{aligned}
&\frac{1}{2T^2}\langle\partial_{x_i}u_j C_{ijkl}\partial_{x_k}u_l\rangle \\
=&\frac{1}{2T^2}(E_{ij}C^{hom}_{ijkl}E_{kl} + \epsilon^2 K_{ijk}F_{ijklmn}K_{mnl} + \epsilon^2(E_{ij}G_{ijklmn}\partial_{Y_l}K_{mlk} \\
&+ \partial_{Y_l}K_{ijk}G_{ijklmn}E_{mn}) + \epsilon^4\partial_{Y_l}K_{ijk}H_{ijklmnop}\partial_{Y_m}K_{opn})
\end{aligned} \tag{21}$$

où

$$F_{ijklmn} = \langle A^{\nabla^2}_{pqkji}C_{pqrs}A^{\nabla^2}_{srlmn}\rangle = \langle B^{\nabla^2}_{pqkji}S_{pqrs}B^{\nabla^2}_{srlmn}\rangle \tag{22}$$

$$G_{ijklmn} = \langle A^{\nabla}_{pqji}C_{pqks}u^{\nabla^2}_{slmn}\rangle = \langle B^{\nabla}_{ijks}u^{\nabla^2}_{slmn}\rangle \tag{23}$$

$$H_{ijklmnop} = \langle u^{\nabla^2}_{skji}C_{slmr}u^{\nabla^2}_{rnop}\rangle \tag{24}$$

les formes des tenseur de localisation des contraintes du premier ordre et du deuxième ordre sont présentées dans les équation (11) et (18). Lorsqu'on fait une homogénéisation pour le milieu périodique, alors le problème est bien posé sur le domaine Y, alors nous pouvons faire une intégration par partie par rapport aux coordonnées macroscopiques. De plus, les termes aux bords sont ignorées. alors on a :

$$\begin{aligned}
&\frac{1}{2T^2}\epsilon E_{ij}\langle A^{\nabla}_{pqji}C_{pqks}u^{\nabla^2}_{slmn}\rangle\partial_{Y_l}K_{mlk} \\
=&\frac{1}{2T^2}\epsilon\int_Y E_{ij}A^{\nabla}_{pqji}C_{pqks}u^{\nabla^2}_{slmn}\partial_{Y_l}K_{mlk} \\
=&-\frac{1}{2T^2}\epsilon\int_Y K_{ijk}A^{\nabla}_{pqji}C_{pqks}u^{\nabla^2}_{slmn}K_{mnl} \\
=&-\frac{1}{2T^2}\epsilon K_{ijk}\langle A^{\nabla}_{pqji}C_{pqks}u^{\nabla^2}_{slmn}\rangle K_{mnl}
\end{aligned}$$

alors on obtient

$$E_{ij}G_{ijklmn}E_{mn,kl} = -E_{ij,k}G_{ijklmn}E_{mn,l}$$
$$E_{ij,kl}G_{ijklmn}E_{mn} = -E_{ij,k}G_{nmlkji}E_{mn,l}$$

Donc l'équation (21) a une forme simplifiée :

$$\begin{aligned}&\frac{1}{2T^2}\langle\partial_{x_i}u_jC_{ijkl}\partial_{x_k}u_l\rangle\\ =&\frac{1}{2T^2}(E_{ij}C^{hom}_{ijkl}E_{kl}+\epsilon^2E_{ij,k}(F_{ijklmn}-G_{ijklmn}-G_{nmlkji})E_{mn,l}+\\ &\epsilon^4E_{ij,kl}H_{ijklmnop}E_{op,nm})\end{aligned} \tag{25}$$

Notons $J_{ijklmn} = F_{ijklmn} - G_{ijklmn} - G_{nmlkji}$.

3 Implémentation

Dans notre projet, nous nous intéressons à la déformation plane dans un milieu périodique, dont la loi de comportement est linéaire, qui peut être écrit sous une forme matricielle.

3.1 Notations matricielles des tenseurs

Dans la théorie d'homogénéisation en gradient d'ordre supérieur, les tenseurs C^{hom}, D, F, G, H sont calculés par des valeurs moyennes d'opérations sur les tenseurs. Afin de mettre en œuvre ces calculs numériques plus adaptés à la code de Python et les subroutines qui seront écrites en Fortran 77. Nous proposons les formes matricielles pour ces différents tenseurs.

Rappel de la notation ingénieur et de Voigt

L'objectif de cette partie est de rappeler les notations de Voigt. Mais auparavant, il est utile de rappeler les notation ingénieur pour la présentation matricielle des tenseurs d'élasticité.

Le principe de la notation ingénieur est bien illustré par le cas des tenseurs symétriques de rang 2 tels que le tenseur des contraintes ou le tenseur des déformations. Pour les tenseurs symétriques de contrainte (σ_{ij}) et de déformation (ϵ_{ij}) par respectivement, les deux colonnes $[\sigma]$ et $[\varepsilon]$ de dimension 3 avec :

$$[\sigma] = \begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} \quad , \quad [\epsilon] = \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ 2\varepsilon_{12}\end{bmatrix}$$

L'introduction du facteur 2 devant ϵ_{12} permet d'assurer l'égalité du produit scalaire $[\sigma]\cdot[\epsilon]$ et du produit contracté $(\sigma_{ij}) : (\epsilon_{ij})$, on a :

$$[\sigma]\cdot[\varepsilon] = (\sigma_{ij}) : (\varepsilon_{ij}) = \sigma_{11}\varepsilon_{11} + \sigma_{22}\varepsilon_{22} + 2\sigma_{12}\varepsilon_{12} \tag{26}$$

alors on établit facilement que la relation de comportement peut s'écrire sous une forme matricielle suivante :

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \Longleftrightarrow [\sigma] = [C]\cdot[\varepsilon]$$

où $[C]$ est la matrice 3×3, symétrique, donnée en fonction des C_{ijkl} par :

$$[C] = \begin{bmatrix}C_{1111} & C_{1122} & C_{1112}\\ C_{2211} & C_{2222} & C_{2212}\\ C_{1211} & C_{1222} & C_{1212}\end{bmatrix} \tag{27}$$

dans laquelle C_{1112}, C_{2212}, C_{1211} et C_{1222} sont nuls en cas de la déformation plane.

Ce genre de notation est utilisé dans Abaqus, en effet la liste d'output de la déformation est ε_{11}, ε_{22}, ε_{33} et $2\varepsilon_{12}$. Mais nous devons faire plusieurs post-traitement après le calcul du premier ordre, afin de trouver la précontrainte et les forces volumiques aux points d'intégration pour le calcul du deuxième ordre et le facteur 2 après ε_{12} nous empêche souvent. Pour faciliter nos calcul, au lieu d'utiliser la notation ingénieur, nous utilisons la notation de Voigt pour la construction de la plupart des tenseurs suivants. Le principe de notation de Voigt est comme la notation ingénieur, mais la construction des deux colonnes $[\sigma]$ et $[\varepsilon]$ sont un peu différent au facteur devant la terme σ_{12} et ε_{12} :

$$[\sigma] = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sqrt{2}\sigma_{12} \end{bmatrix} \quad , \quad [\varepsilon] = \begin{bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \sqrt{2}\varepsilon_{12} \end{bmatrix}$$

cette notation aussi assure l'égalité du produit scalaire des matrices et du produit contracté entre le tenseur de contrainte et de déformation, qui est déjà présentée dans l'équation (26). Dans ce cas là, la relation peut s'écire sous la forme matricielle suivante :

$$\sigma_{ij} = C_{ijkl}\varepsilon_{kl} \Longleftrightarrow [\sigma] = [C] \cdot [\varepsilon]$$

où la matrice $[C]$ possède des coefficients un peu différent à la notation ingénieur :

$$[C] = \begin{bmatrix} C_{1111} & C_{1122} & \sqrt{2}C_{1112} \\ C_{2211} & C_{2222} & \sqrt{2}C_{2212} \\ \sqrt{2}C_{1211} & \sqrt{2}C_{1222} & 2C_{1212} \end{bmatrix} \tag{28}$$

dans laquelle C_{1112}, C_{2212}, C_{1211} et C_{1222} sont nuls si le matériau présent un axe de symétrie.

Notations matricielles adaptées

Notations matricielles de (E_{ij}) et (K_{ijk}) Premièrement, nous faisons un accord pour la notation du vecteur de déformation macroscopique (E_{ij}) et de second gradient macroscopique $E_{ij,k} = K_{ijk}$, les deux colonnes $[E]$ et $[K]$ sont illustrées de dimension 3 et 6, respectivement avec :

$$[E] = \begin{bmatrix} E_{11} \\ E_{22} \\ \sqrt{2}E_{12} \end{bmatrix} \quad , \quad [K] = \begin{bmatrix} K_{111} \\ K_{221} \\ \sqrt{2}K_{121} \\ K_{112} \\ K_{222} \\ \sqrt{2}K_{122} \end{bmatrix} \tag{29}$$

Notations matricielles de (u^{∇}_{ijk}) et $(u^{\nabla^2}_{ijkl})$ Pour calculer $[C^{hom}]$, $[D]$, $[F]$, $[G]$ et $[H]$, nous devons premièrement calculer les tenseurs $(u^{\nabla}_{ijk})(y_q)$ et $(u^{\nabla^2}_{ijkl})(y_q)$ et les écrivons selon les notations matricielles $[u^{\nabla}](y_q)]$ et $[u^{\nabla^2}](y_q)$. Les formes matricielles des tenseur $(u^{\nabla}_{ijk})(y_q)$ et $(u^{\nabla}_{ijkl})(y_q)$ peuvent être écrits de dimension 2×3 et 2×6, respectivement :

$$\begin{aligned} &[u^{\nabla}](y_q) \\ &= \begin{bmatrix} u^{\nabla}_{111}(y_q) & u^{\nabla}_{122}(y_q) & \sqrt{2}u^{\nabla}_{112}(y_q) \\ u^{\nabla}_{211}(y_q) & u^{\nabla}_{222}(y_q) & \sqrt{2}u^{\nabla}_{212}(y_q) \end{bmatrix} \end{aligned} \tag{30}$$

$$
\begin{aligned}
&[u^{\nabla^2}](y_q) \\
&= \begin{bmatrix} u^{\nabla^2}_{1111}(y_q) & u^{\nabla^2}_{1122}(y_q) & \sqrt{2}u^{\nabla^2}_{1121}(y_q) & u^{\nabla^2}_{1211}(y_q) & u^{\nabla^2}_{1222}(y_q) & \sqrt{2}u^{\nabla^2}_{1221}(y_q) \\ u^{\nabla^2}_{2111}(y_q) & u^{\nabla^2}_{2122}(y_q) & \sqrt{2}u^{\nabla^2}_{2121}(y_q) & u^{\nabla^2}_{2211}(y_q) & u^{\nabla^2}_{2222}(y_q) & \sqrt{2}u^{\nabla^2}_{2221}(y_q) \end{bmatrix}
\end{aligned} \tag{31}
$$

Et la forme matricielle du tenseur de localisation des contraintes (B^{∇}_{ijkl}) peut être écrit en dimension de 2×6 :

$$
\begin{aligned}
&[B^{\nabla}](y_q) \\
&= \begin{bmatrix} B^{\nabla}_{1111}(y_q) & B^{\nabla}_{1122}(y_q) & \sqrt{2}B^{\nabla}_{1121}(y_q) & B^{\nabla}_{1211}(y_q) & B^{\nabla^2}_{1222}(y_q) & \sqrt{2}B^{\nabla}_{1221}(y_q) \\ B^{\nabla}_{2111}(y_q) & B^{\nabla}_{2122}(y_q) & \sqrt{2}B^{\nabla}_{2121}(y_q) & B^{\nabla}_{2211}(y_q) & B^{\nabla}_{2222}(y_q) & \sqrt{2}B^{\nabla}_{2221}(y_q) \end{bmatrix} \\
&= \begin{bmatrix} \sigma^{E_{11}}_{11} & \sigma^{E_{22}}_{11} & \sqrt{2}\sigma^{E_{12}}_{11} & \sigma^{E_{11}}_{12} & \sigma^{E_{22}}_{12} & \sqrt{2}\sigma^{E_{12}}_{12} \\ \sigma^{E_{11}}_{12} & \sigma^{E_{22}}_{12} & \sqrt{2}\sigma^{E_{12}}_{12} & \sigma^{E_{11}}_{22} & \sigma^{E_{22}}_{22} & \sqrt{2}\sigma^{E_{12}}_{22} \end{bmatrix} \\
&= \begin{bmatrix} \sigma^{E_{11}}_{11} & \sigma^{E_{22}}_{11} & \sigma^{\sqrt{2}E_{12}}_{11} & \sigma^{E_{11}}_{12} & \sigma^{E_{22}}_{12} & \sigma^{\sqrt{2}E_{12}}_{12} \\ \sigma^{E_{11}}_{12} & \sigma^{E_{22}}_{12} & \sigma^{\sqrt{2}E_{12}}_{12} & \sigma^{E_{11}}_{22} & \sigma^{E_{22}}_{22} & \sigma^{\sqrt{2}E_{12}}_{22} \end{bmatrix}
\end{aligned} \tag{32}
$$

Dans laquelle, quelque soit la variable X, $\sqrt{2}X^{E_{12}} = X^{\sqrt{2}E_{12}}$ est assuré selon la proposition de la linéarité de la loi de comportement.

$[u^{\nabla}]$ et $[u^{\nabla^2}]$ sont plus délicates à calculer. D'abord, nous écrivons la colonne des déplacement du problème du premier ordre u^1_i à l'aide de l'équation (5) en ôtant la valeur de $E^0_{ij}y_j$:

$$
\begin{bmatrix} u^1_1 \\ u^1_2 \end{bmatrix} = \begin{bmatrix} u^{Aba1}_1 \\ u^{Aba1}_2 \end{bmatrix} - \begin{bmatrix} y^{E_{11}}_1 & 0 & \dfrac{y^{E_{12}}_2}{\sqrt{2}} \\ 0 & y^{E_{22}}_2 & \dfrac{y^{E_{12}}_2}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} E_{11} \\ E_{22} \\ \sqrt{2}E_{12} \end{bmatrix}
$$

Dans laquelle, u^{Aba1}_i est l'inconnu que nous devons résoudre dans le problème de premier ordre. y_i est la coordonnée par rapport au point référent qu'on supposera immobile dans toutes les directions pendant l'implémentation.

$$
\begin{bmatrix} u^{Aba1}_1 \\ u^{Aba1}_2 \end{bmatrix} = \begin{bmatrix} u^{E_{11}}_1 & u^{E_{22}}_1 & u^{\sqrt{2}E_{12}}_1 \\ u^{E_{11}}_2 & u^{E_{22}}_2 & u^{\sqrt{2}E_{12}}_2 \end{bmatrix} \begin{bmatrix} E_{11} \\ E_{22} \\ \sqrt{2}E_{12} \end{bmatrix}
$$

or on peut écrire l'équation (4) sous une forme matricielle :

$$
\begin{bmatrix} u^{\nabla}_{111}(y_q) & u^{\nabla}_{122}(y_q) & \sqrt{2}u^{\nabla}_{112}(y_q) \\ u^{\nabla}_{211}(y_q) & u^{\nabla}_{222}(y_q) & \sqrt{2}u^{\nabla}_{212}(y_q) \end{bmatrix} [E] = \begin{bmatrix} u^1_1 \\ u^1_2 \end{bmatrix} - \langle \begin{bmatrix} u^1_1 \\ u^1_2 \end{bmatrix} \rangle
$$

alors on obtient un calcul matricielle pour chercher $[u^{\nabla}]$:

$$
\begin{aligned}
[u^{\nabla}] &= \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{122} & \sqrt{2}u^{\nabla}_{112} \\ u^{\nabla}_{211} & u^{\nabla}_{222} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \\
&= \begin{bmatrix} u^{E_{11}}_1 & u^{E_{22}}_1 & u^{\sqrt{2}E_{12}}_1 \\ u^{E_{11}}_2 & u^{E_{22}}_2 & u^{\sqrt{2}E_{12}}_2 \end{bmatrix} - \begin{bmatrix} y^{E_{11}}_1 & 0 & \dfrac{y^{E_{12}}_2}{\sqrt{2}} \\ 0 & y^{E_{22}}_2 & \dfrac{y^{E_{12}}_2}{\sqrt{2}} \end{bmatrix} - [U^{\nabla}_1]
\end{aligned} \tag{33}
$$

dans laquelle on a :

$$[U_1^{\nabla}] = \begin{bmatrix} U_1^{E_{11}} & U_1^{E_{22}} & U_1^{\sqrt{2}E_{12}} \\ U_2^{E_{11}} & U_2^{E_{22}} & U_2^{\sqrt{2}E_{12}} \end{bmatrix}$$
$$= \left\langle \begin{bmatrix} u_1^{E_{11}} & u_1^{E_{22}} & u_1^{\sqrt{2}E_{12}} \\ u_2^{E_{11}} & u_2^{E_{22}} & u_2^{\sqrt{2}E_{12}} \end{bmatrix} - \begin{bmatrix} y_1^{E_{11}} & 0 & \frac{y_2^{E_{12}}}{\sqrt{2}} \\ 0 & y_2^{E_{22}} & \frac{y_2^{E_{12}}}{\sqrt{2}} \end{bmatrix} \right\rangle \tag{34}$$

Pour calculer $[u^{\nabla^2}]$, on le calcul directement en ôtant la valeur moyenne des déplacement de la cellule dans les étapes différentes :

$$[u^{\nabla^2}] = \begin{bmatrix} u_1^{K_{111}} & u_1^{K_{221}} & u_1^{\sqrt{2}K_{121}} & u_1^{K_{112}} & u_1^{K_{222}} & u_1^{\sqrt{2}K_{122}} \\ u_2^{K_{111}} & u_2^{K_{221}} & u_2^{\sqrt{2}K_{121}} & u_2^{K_{112}} & u_2^{K_{222}} & u_2^{\sqrt{2}K_{122}} \end{bmatrix} - [U_2^{\nabla^2}] \tag{35}$$

dans laquelle on a :

$$[U_2^{\nabla^2}] = \begin{bmatrix} \langle u_1^{K_{111}} \rangle & \langle u_1^{K_{221}} \rangle & \langle u_1^{\sqrt{2}K_{121}} \rangle & \langle u_1^{K_{112}} \rangle & \langle u_1^{K_{222}} \rangle & \langle u_1^{\sqrt{2}K_{122}} \rangle \\ \langle u_2^{K_{111}} \rangle & \langle u_2^{K_{221}} \rangle & \langle u_2^{\sqrt{2}K_{121}} \rangle & \langle u_2^{K_{112}} \rangle & \langle u_2^{K_{222}} \rangle & \langle u_2^{\sqrt{2}K_{122}} \rangle \end{bmatrix} \tag{36}$$

Notations matricielles de (C_{ijkl}^{hom}), (D_{ijklmn}), (F_{ijklmn}), (F_{ijklmn}) et $(H_{ijklmnop})$ Nous pouvons écrire les équations (14) et (25) qui expriment l'énergie de la déformation dans les problème du premier ordre et deuxième ordre selon la notation de Voigt. Liant aux notation de voigt pour $[E]$ et $[K]$, alors $[C^{hom}]$ est illustré de dimension 3×3 :

$$[C^{hom}] = \begin{bmatrix} C_{1111}^{hom} & C_{1122}^{hom} & \sqrt{2}C_{1112}^{hom} \\ C_{2211}^{hom} & C_{2222}^{hom} & \sqrt{2}C_{2212}^{hom} \\ \sqrt{2}C_{1211}^{hom} & \sqrt{2}C_{1222}^{hom} & 2C_{1212}^{hom} \end{bmatrix} \tag{37}$$

$[D]$, $[F]$, $[G]$ sont de dimension 6×6, respectivement. Les indices des coefficients de la matrice sont dans l'équation (38).

$$\begin{bmatrix} D_{111111} & D_{111122} & \sqrt{2}D_{111121} & D_{111211} & D_{111222} & \sqrt{2}D_{111221} \\ D_{221111} & D_{221122} & \sqrt{2}D_{221121} & D_{221211} & D_{221222} & \sqrt{2}D_{221221} \\ \sqrt{2}D_{121111} & \sqrt{2}D_{121122} & 2D_{121121} & \sqrt{2}D_{121211} & \sqrt{2}D_{121222} & 2D_{121221} \\ D_{112111} & D_{112122} & \sqrt{2}D_{112121} & D_{112211} & D_{112222} & \sqrt{2}D_{112221} \\ D_{222111} & D_{222122} & \sqrt{2}D_{222121} & D_{222211} & D_{222222} & \sqrt{2}D_{222221} \\ \sqrt{2}D_{122111} & \sqrt{2}D_{122122} & 2D_{122121} & \sqrt{2}D_{122211} & \sqrt{2}D_{122222} & 2D_{122221} \end{bmatrix} \tag{38}$$

Nous écrivons ces trois matrices par blocs pour comprendre le calcul par blocs qui sera indiqué ultérieurement.

$$[D] = \begin{bmatrix} D_{ij11kl} & D_{ij12kl} \\ D_{ij21kl} & D_{ij22kl} \end{bmatrix} \tag{39}$$

$$[F] = \begin{bmatrix} F_{ij11kl} & F_{ij12kl} \\ F_{ij21kl} & F_{ij22kl} \end{bmatrix} \tag{40}$$

$$[G] = \begin{bmatrix} G_{ij11kl} & G_{ij12kl} \\ G_{ij21kl} & G_{ij22kl} \end{bmatrix} \tag{41}$$

Dans lesquelles, les sous-matrices, par exemples D_{ij11kl}, F_{ij11kl} et G_{ij11kl}, sont tous de dimension 3×3. Différent aux $[D]$, $[F]$, $[G]$, la matrice $[H]$ est de dimension 12×12 :

$$[H] = \begin{bmatrix} H_{ijk11nop} & H_{ijk12nop} \\ H_{ijk21nop} & H_{ijk22nop} \end{bmatrix} \tag{42}$$

Les notations des matrices par blocs présentées dans l'équation (39), (40), (41) et (42) sont commodes à utiliser en faisant les multiplications, Par exemple, afin de calculer D_{ij11kl}, D_{ij12kl}, D_{ij21kl}, D_{ij22kl} à l'aide de l'équation (15), on obtient :

$$
\begin{aligned}
D_{ij11kl} &= \langle [u^{\nabla}]^t(y_q) \begin{bmatrix} C_{1111}(y_q) & C_{1112}(y_q) \\ C_{2111}(y_q) & C_{2112}(y_q) \end{bmatrix} [u^{\nabla}](y_q) \rangle \\
&= \langle \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{211} \\ u^{\nabla}_{122} & u^{\nabla}_{222} \\ \sqrt{2}u^{\nabla}_{112} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \begin{bmatrix} C_{1111} & 0 \\ 0 & C_{1212} \end{bmatrix} \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{122} & \sqrt{2}u^{\nabla}_{112} \\ u^{\nabla}_{211} & u^{\nabla}_{222} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \rangle \\
D_{ij12kl} &= \langle [u^{\nabla}]^t(y_q) \begin{bmatrix} C_{1121}(y_q) & C_{1122}(y_q) \\ C_{2121}(y_q) & C_{2122}(y_q) \end{bmatrix} [u^{\nabla}](y_q) \rangle \\
&= \langle \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{211} \\ u^{\nabla}_{122} & u^{\nabla}_{222} \\ \sqrt{2}u^{\nabla}_{112} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \begin{bmatrix} 0 & C_{1122} \\ C_{1212} & 0 \end{bmatrix} \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{122} & \sqrt{2}u^{\nabla}_{112} \\ u^{\nabla}_{211} & u^{\nabla}_{222} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \rangle \\
D_{ij21kl} &= \langle [u^{\nabla}]^t(y_q) \begin{bmatrix} C_{1211}(y_q) & C_{1212}(y_q) \\ C_{2211}(y_q) & C_{2212}(y_q) \end{bmatrix} [u^{\nabla}](y_q) \rangle \\
&= \langle \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{211} \\ u^{\nabla}_{122} & u^{\nabla}_{222} \\ \sqrt{2}u^{\nabla}_{112} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \begin{bmatrix} 0 & C_{1212} \\ C_{2211} & 0 \end{bmatrix} \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{122} & \sqrt{2}u^{\nabla}_{112} \\ u^{\nabla}_{211} & u^{\nabla}_{222} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \rangle \\
D_{ij22kl} &= \langle [u^{\nabla}]^t(y_q) \begin{bmatrix} C_{1221}(y_q) & C_{1222}(y_q) \\ C_{2221}(y_q) & C_{2222}(y_q) \end{bmatrix} [u^{\nabla}](y_q) \rangle \\
&= \langle \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{211} \\ u^{\nabla}_{122} & u^{\nabla}_{222} \\ \sqrt{2}u^{\nabla}_{112} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \begin{bmatrix} C_{1212} & 0 \\ 0 & C_{2222} \end{bmatrix} \begin{bmatrix} u^{\nabla}_{111} & u^{\nabla}_{122} & \sqrt{2}u^{\nabla}_{112} \\ u^{\nabla}_{211} & u^{\nabla}_{222} & \sqrt{2}u^{\nabla}_{212} \end{bmatrix} \rangle
\end{aligned}
$$

De la même façon matricielle pour calculer les sous-matrices de $[G]$ à partir de l'équation (23) :

$$
\begin{aligned}
G_{ij11kl} &= \langle [B^{\nabla}]^t(y_q) \begin{bmatrix} C_{1111} & 0 \\ 0 & C_{1212} \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
G_{ij12kl} &= \langle [B^{\nabla}]^t(y_q) \begin{bmatrix} 0 & C_{1122} \\ C_{1212} & 0 \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
G_{ij21kl} &= \langle [B^{\nabla}]^t(y_q) \begin{bmatrix} 0 & C_{1212} \\ C_{2211} & 0 \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
G_{ij22kl} &= \langle [B^{\nabla}]^t(y_q) \begin{bmatrix} C_{1212} & 0 \\ 0 & C_{2222} \end{bmatrix} [u^{\nabla^2}](y_q) \rangle
\end{aligned}
$$

et de $[H]$ à partir de l'équation (24) :

$$
\begin{aligned}
H_{ijk11nop} &= \langle [u^{\nabla^2}]^t(y_q) \begin{bmatrix} C_{1111} & 0 \\ 0 & C_{1212} \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
H_{ijk12nop} &= \langle [u^{\nabla^2}]^t(y_q) \begin{bmatrix} 0 & C_{1122} \\ C_{1212} & 0 \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
H_{ijk21nop} &= \langle [u^{\nabla^2}]^t(y_q) \begin{bmatrix} 0 & C_{1212} \\ C_{2211} & 0 \end{bmatrix} [u^{\nabla^2}](y_q) \rangle \\
H_{ijk22nop} &= \langle [u^{\nabla^2}]^t(y_q) \begin{bmatrix} C_{1212} & 0 \\ 0 & C_{2222} \end{bmatrix} [u^{\nabla^2}](y_q) \rangle
\end{aligned}
$$

3.2 Effet des symétries matricielles de la cellule unitaire

Dans l'application numérique, nous nous intéressons souvent à lancer la simulation numérique dans une cellule qui possède deux plans symétriques perpendiculaires. Alors afin d'économiser le nombre de maillages, nous faisons souvent la simulation dans un quart de la cellule (la partie en rouge dans la figure 1). Dans ce cas, l'effet symétrique grâce aux conditions aux bords est subtile. Lorsqu'on prend un quart de la cellule, nous supposons que le plan "Gauche" et "Bas" sont les deux plans géométriquement symétriques perpendiculaires.

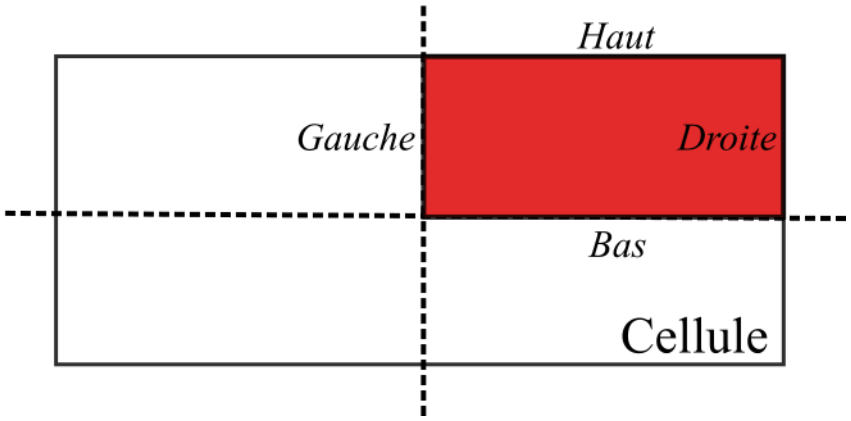

FIGURE 1 – Symétrie géométrique de la cellule

Parité des champs $[u^{\nabla}]$ et $[u^{\nabla^2}]$

On présente la relation symétrique et antisymétrique des étapes par rapport aux deux plans géométriquement symétrique de la cellule dans le problème du premier ordre et du deuxième ordre dans le tableau 1 :

Opérateur de symétrie	$\mathrm{P} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\mathrm{P} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$
Problème du premier ordre		
E_{11}	$+$	$+$
E_{22}	$+$	$+$
$\sqrt{2}E_{12}$	$-$	$-$
Problème du deuxième ordre		
K_{111}	$-$	$+$
K_{221}	$-$	$+$
$\sqrt{2}K_{121}$	$+$	$-$
K_{112}	$+$	$-$
K_{222}	$+$	$-$
$\sqrt{2}K_{122}$	$-$	$+$

$+$: symétrique $\quad$ $-$: antisymétrique

Tableau 1 – Relation de symétrie du déplacement dans les étapes différentes

Dans le tableau 1, nous voyons bien la relation symétrique du déplacement dans le quart de la cellule, Par exemple, Pour la traction dans les deux directions E_{11} et E_{22}, les déplacements

sont symétriques par rapport aux plans “Gauche” et “Bas” :

$$\begin{aligned} u_1(y_1, y_2) &= u_1(y_1, -y_2) \\ u_2(y_1, y_2) &= -u_2(y_1, -y_2) \\ u_1(y_1, y_2) &= -u_1(-y_1, y_2) \\ u_2(y_1, y_2) &= u_2(-y_1, y_2) \end{aligned}$$

alors dans ce cas, les valeurs moyennes du déplacement dans les quatre partie de la cellule sont calculées, respectivement :

$$\begin{aligned} \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_1(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_1(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1<0,y_2>0} u_1(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} -u_1(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1>0,y_2<0} u_1(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_1(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1<0,y_2<0} u_1(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} -u_1(y_1, y_2)dy_1dy_2 \end{aligned} \tag{43}$$

alors la moyenne du déplacement dans la cellule dans la direction 1 est nulle (la somme des quatre équations de (43)) et la moyenne du déplacement dans le quart de la cellule (la partie en rouge dans la figure 1) n'a pas la peine d'être nulle.

nous trouvons que la moyenne du déplacement dans la direction 2 est aussi nulle car on a les quatre équations de (44) :

$$\begin{aligned} \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_2(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_2(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1<0,y_2>0} u_2(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} u_2(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1>0,y_2<0} u_2(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} -u_2(y_1, y_2)dy_1dy_2 \\ \frac{1}{l_1 l_2}\int_{y_1<0,y_2<0} u_2(y_1, y_2)dy_1dy_2 &= \frac{1}{l_1 l_2}\int_{y_1>0,y_2>0} -u_2(y_1, y_2)dy_1dy_2 \end{aligned} \tag{44}$$

Et Pour le cisaillement $\sqrt{2}E_{12}$, les déplacements sont antisymétriques par rapport aux plans “Gauche” et “Bas”, alors on a :

$$\begin{aligned} u_1(y_1, y_2) &= -u_1(y_1, -y_2) \\ u_2(y_1, y_2) &= u_2(y_1, -y_2) \\ u_1(y_1, y_2) &= u_1(-y_1, y_2) \\ u_2(y_1, y_2) &= -u_2(-y_1, y_2) \end{aligned}$$

En calculant la somme des énergies moyennes dans les quatre parties d'une cellule, nous trouvons aussi que les déplacements moyens dans la direction 1 et 2 sont nuls.

A partir des preuves avant, on a vu que les déplacements moyens de la cellule ne sont pas toujours égaux aux ceux dans le quart de cellule (en rouge dans la figure 1). Si on fait nos calculs dans un quart de la cellule, alors dans l'équation (34), les coefficients dans la matrice de $[U_1^{\nabla}]$ sont nulles au lieu de mettre les déplacements moyens de la partie en rouge. C'est-à-dire nous n'avons pas besoin d'ôter la valeur moyenne des trois étapes quand on calcule $[u^{\nabla}]$

dans un quart de la cellule. Nous revenons à l'équation (33), et les coefficients sont modifiés en mettant $U_1^{E_{11}} = U_2^{E_{11}} = U_1^{E_{22}} = U_2^{E_{22}} = U_1^{\sqrt{2}E_{12}} = U_2^{\sqrt{2}E_{12}} = 0$:

$$[U_1^{\nabla}] = \begin{bmatrix} U_1^{E_{11}} & U_1^{E_{22}} & U_1^{\sqrt{2}E_{12}} \\ U_2^{E_{11}} & U_2^{E_{22}} & U_2^{\sqrt{2}E_{12}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

et pour les déplacements moyens de la cellule dans le problème du deuxième ordre sont antisymétriques au plan "Bas" pour les second gradients K_{111}, K_{221} et K_{122}, et au plan "Gauche" pour les second gradients $_{K121}$, K_{112} et K_{222}, alors en calculant le $[u^{\nabla^2}]$ dans l'équation (35), nous n'avons pas besoin d'ôter le déplacement moyen dans la direction 1 pour les second gradients K_{121}, K_{112} et K_{222} et dans la direction 2 pour les second gradients K_{111}, K_{221} et K_{122}, puisque dans ce cas, par exemple, pour le second gradient K_{111}, on a :

$$\begin{aligned} u_1(y_1, x_2) &= -u_1(y_1, -y_2) \\ u_2(y_1, x_2) &= u_2(y_1, -y_2) \\ u_1(x_1, x_2) &= -u_1(-x_1, x_2) \\ u_2(x_1, x_2) &= u_2(-x_1, x_2) \end{aligned}$$

Par la même manière d'analyse, nous trouvons facilement que la matrice $[U_2^{\nabla^2}]$ dans l'équation (36) s'écrit dans ce cas à partir de l'équation :

$$[U_2^{\nabla^2}] = \begin{bmatrix} \langle u_1^{K_{111}} \rangle & \langle u_1^{K_{221}} \rangle & 0 & 0 & 0 & \langle u_1^{\sqrt{2}K_{122}} \rangle \\ 0 & 0 & \langle u_2^{\sqrt{2}K_{121}} \rangle & \langle u_2^{K_{112}} \rangle & \langle u_2^{K_{222}} \rangle & 0 \end{bmatrix}$$

C'est-à-dire, par exemple, la valeur moyenne de $u_1^{K_{111}}$ dans un quart de la cellule est identique à celle dans la cellule totale.

Forme de $[C^{hom}]$, $[D]$, $[F]$, $[G]$ et $[H]$

Lorsque'on fait le calcul de $[C^{hom}]$, $[D]$, $[F]$, $[G]$ et $[H]$ en faisant une multiplication des matrices, nous trouvons normalement tous les coefficients sont non nuls dans le calcul numérique. Mais n'oublie pas que les propriétés de la déformation plane et les parités du champs soient respectés dans ces calculs. Par exemple, pour $[C^{hom}]$, lorsqu'on calcul dans un modèle de la déformation plane, alors les termes $C^{hom}_{123\alpha}$ sont tous nuls, alors en effet la forme de $[C^{hom}]$ de la notation de Voigt dans l'équation (37) s'écrit comme :

$$[C^{hom}] = \begin{bmatrix} C^{hom}_{1111} & C^{hom}_{1122} & 0 \\ C^{hom}_{2211} & C^{hom}_{2222} & 0 \\ 0 & 0 & 2C^{hom}_{1212} \end{bmatrix}$$

de la même raison, $[D]$, $[F]$, $[G]$ ont les coefficients nuls dans leurs matrices, présenté dans l'équation (45) car l'antisymétrie de $[u^{\nabla}]$ et $[u^{\nabla^2}]$:

$$\begin{bmatrix} D_{111111} & D_{111122} & 0 & 0 & 0 & \sqrt{2}D_{111221} \\ D_{221111} & D_{221122} & 0 & 0 & 0 & \sqrt{2}D_{221221} \\ 0 & 0 & 2D_{121121} & \sqrt{2}D_{121211} & \sqrt{2}D_{121222} & 0 \\ 0 & 0 & \sqrt{2}D_{112121} & D_{112211} & D_{112222} & 0 \\ 0 & 0 & \sqrt{2}D_{222121} & D_{222211} & D_{222222} & 0 \\ \sqrt{2}D_{212111} & \sqrt{2}D_{212122} & 0 & 0 & 0 & 2D_{212212} \end{bmatrix} \tag{45}$$

et $[F]$, $[G]$, ont les coefficients nuls avec les mêmes indices car ils gardent la même propriété antisymétrique avec $[D]$. Pour le H, les sous-matrices $H_{ijk11nop}$ et $H_{ijk22nop}$ ont la même forme

comme $[D]$. En revanche, $H_{ijk12nop}$ et $H_{ijk21nop}$ ont une forme opposée par rapport à $[D]$. Pour illustrer cette différence, la forme de la sous-matrice $H_{ijk12nop}$ est présentée dans l'équation (46) :

$$\begin{bmatrix} 0 & 0 & \sqrt{2}H_{11112121} & H_{11112211} & H_{11112222} & 0 \\ 0 & 0 & \sqrt{2}H_{22112121} & H_{22112211} & H_{22112222} & 0 \\ \sqrt{2}H_{12112111} & \sqrt{2}H_{12112122} & 0 & 0 & 0 & 2H_{12112221} \\ H_{11212111} & H_{11212122} & 0 & 0 & 0 & \sqrt{2}H_{11212221} \\ H_{22212111} & H_{22212122} & 0 & 0 & 0 & \sqrt{2}H_{22212221} \\ 0 & 0 & 2H_{12212121} & \sqrt{2}H_{12212211} & \sqrt{2}H_{12212222} & 0 \end{bmatrix} \tag{46}$$

Dans la discussions après, nous notons que $D_{11} = D_{111111}$, $D_{12} = D_{111122}$ etc pour simplifier les indices des notations.

3.3 Implémentation dans Abaqus

Généralité

ABAQUS est un logiciel de calcul d'éléments finis développé par ABAQUS, Inc (Dassault Systèmes). Il se compose de trois produits : ABAQUS/Standard, ABAQUS/Explicit et ABAQUS/CAE :

- ABAQUS/Standard est un solveur généraliste qui recourt à un schéma traditionnel d'intégration implicite.
- Le solveur ABAQUS/Explicit emploie un schéma d'intégration explicite pour résoudre des problèmes dynamiques ou quasi statiques non linéaires.
- ABAQUS/CAE constitue une interface intégrée de visualisation et de modélisation pour lesdits solveurs.

Pendant notre projet, nous utilisons ABAQUS/Standard pour résoudre notre problème. Au lieu de profiter de l'interface CAE, nous faisons notre modèle géométrique, les maillages et la configuration du travail (job en anglais) à l'aide de Python et Fortran 77 et les résultats sont visualisés par Abaqus/CAE Kernel. Le processus du calcul est décrit dans la figure 2.

Dans Abaqus, nous décrivons d'abord le modèle géométrique et les propriétés des maillages et puis créons les étapes différentes pour faire la simulation numérique. Pendant la configuration d'étape du calcul, nous donnons les conditions aux limites et les champs pré-définis s'ils sont demandés. Abaqus va automatiquement lancer la simulation et stocker les résultats numériquement dans un fichier .odb.

Type d'éléments

En faisant le calcul numérique, il y a plein de choix de l'élément. Pour un modèle de déformation plane, Voici quelques choix du type d'éléments usuels dans la figure 3, dans laquelle il inclut l'élément complet et réduit. Par exemple, pour l'élément quadrangulaire, l'élément complet a quatre points d'intégration, pourtant l'élément réduit n'a que un. Pendant notre projet, nous utilisons l'élément CPE4. La lettre "C" dans son nom signifie la déplacement et la contrainte soit continue, les lettres "PE" signifie la déformation plane (plane strain en anglais) et le numéro 4 signifie qu'il y a quatre nœuds en total. Si on veut indiquer l'élément réduit, une lettre "R" est demandée d'ajouter à la fin, comme CPE4R.

Configuration des étapes

Pour un modèle de la contrainte plane du problème de premier ordre, il nous faut définir au mois 3 étapes en statique, perturbations linéaires dans Abaqus. Les valeurs des coefficients

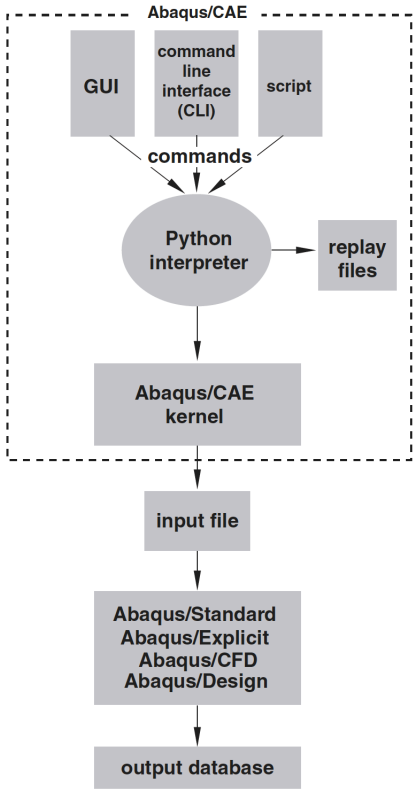

FIGURE 2 – Abaqus Scripting interface et Abaqus/CAE Kernel [1]

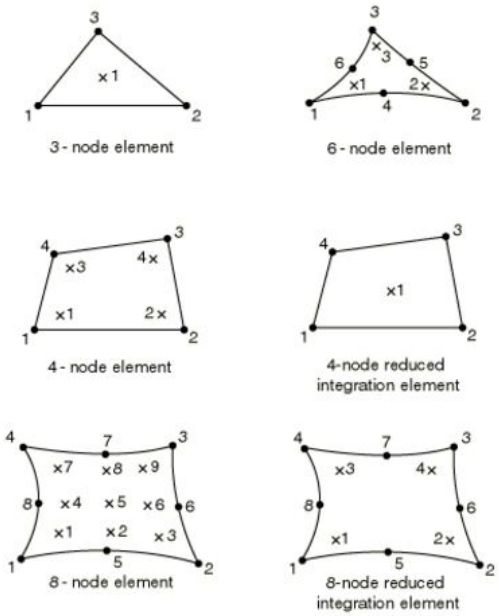

FIGURE 3 – Type d'élément et la numérotation des point d'intégrations associées

Étape	E_{11}	E_{22}	E_{12}
Step-E_{11}	1	0	1
Step-E_{22}	0	1	0
Step-E_{12}	0	0	$1/\sqrt{2}$

Tableau 2 – Les valeurs de E_{ij} dans le problème du premier ordre

Étape	K_{111}	K_{221}	K_{121}	K_{112}	K_{222}	K_{122}
Step-K_{111}	1	0	0	0	0	0
Step-K_{221}	0	1	0	0	0	0
Step-K_{121}	0	0	$1/\sqrt{2}$	0	0	0
Step-K_{112}	0	0	0	1	0	0
Step-K_{222}	0	0	0	0	1	0
Step-K_{122}	0	0	0	0	0	$1/\sqrt{2}$

Tableau 3 – Les valeurs de K_{ijk} dans le problème du deuxième ordre

de $[E]$ et $[K]$ sont présentées qui correspondent à leurs notations de Voigt. Les conditions aux limites des trois étapes du problème du premier ordre sont présentées dans le tableau 2. Et pour un modèle de la contrainte plane du problème de deuxième ordre, il nous faut définir aussi 6 étapes et la matrice $[K]$ est unitaire dans chaque étape. Les valeurs des déformations macroscopiques des six étapes sont présentées dans le tableau 3.

Conditions aux limites

Grâce au tableau 1, 2 et 3. nous avons les valeurs de E_{ij} et K_{ij} pour les étapes qui possèdent une symétrie ou une antisymétrie aux bords. Or le champ du déplacement ξ_i dans l'équation (5) est une fonction périodique aux bords. alors les déplacements concrets liés aux déformation macroscopique E_{ij} est obligatoire d'être mis en œuvre. Par exemple, une traction est demandée selon la direction 1 dans le Step-E_{11}, alors un déplacement dans le plan "Droite" est $u_1 = E_{11}l_1 = l_1$ où l_1 est la moitié de la longueur de la cellule.

étape	Plan	conditions aux limites
STEP-E_{11}	Haut	$u_2 = 0$ et u_1 indéfini
	Bas	$u_2 = 0$ et u_1 indéfini
	Gauche	$u_1 = 0$ et u_2 indéfini
	Droite	$u_1 = l_1$ et u_2 indéfini
STEP-E_{22}	Haut	$u_2 = l_2$ et u_1 indéfini
	Bas	$u_2 = 0$, u_1 indéfini
	Gauche	$u_1 = 0$ et u_2 indéfini
	Droite	$u_1 = 0$ et u_2 indéfini
STEP-E_{12}	Haut	$u_1 = l_2/\sqrt{2}$ et u_2 indéfini
	Bas	$u_1 = 0$ et u_2 indéfini
	Gauche	$u_1 = 0$ et u_1 indéfini
	Droite	$u_2 = l_1/\sqrt{2}$, u_1 indéfini

Tableau 4 – Conditions aux limites dans le problème du premier ordre

Les conditions aux limites dans les plans différents sont présentés dans le tableau 4. Pour le problème du deuxième ordre, Les conditions aux limites contient non seulement la symétrie et l'antisymétrie, mais aussi un point fixé pour éviter les translations globales et assurer l'unicité de la solution numérique. Les conditions aux limites dans problème du deuxième ordre sont présentées dans le tableau 5.

étape	Plan/Point	conditions aux limites
STEP-K_{111}	Haut, Bas, Gauche, Droite	$u_2 = 0$, u_1 indéfini
STEP-K_{221}	Haut, Bas, Gauche, Droite	$u_2 = 0$, u_1 indéfini
STEP-K_{121}	Haut, Bas, Gauche, Droite	$u_1 = 0$, u_2 indéfini
STEP-K_{112}	Haut, Bas, Gauche, Droite	$u_1 = 0$, u_2 indéfini
STEP-K_{222}	Haut, Bas, Gauche, Droite	$u_1 = 0$, u_2 indéfini
STEP-K_{122}	Haut, Bas, Gauche, Droite	$u_2 = 0$, u_1 indéfini
All Steps	Le point O	$u_1 = u_2 = 0$

Tableau 5 – Conditions aux limites dans le problème du deuxième ordre

Dload

Pour implémenter les forces volumiques, nous utilisons la subroutine appelée *DLOAD*. À l'aide de cette subroutine, nous pouvons définir les petites forces aux différentes points d'intégration dans chaque élément qui présentent en effet la force volumique qu'on vient de discuter dans la théorie. Dans l'application de pantographe qui sera présentée après, nous voyons que chaque cellule dans le milieu périodique contient les parties vides, alors il nous faut faire un accord pour les définitions des valeurs moyennes de la cellule. Dans le calcul après, nous définissons que Σ^0_{ij} est la valeur moyenne de la contrainte macroscopique avec les vides dans l'équation (8).

La première équation (16) est correcte même s'il y a d'espace vide dans la cellule. Mais dans ce cas $F_i = -\frac{s_{cell}}{s_{mat}}\partial_{Y_j}\langle\sigma^0_{ij}\rangle$ et pour la partie vide, il n'y pas de force volumique. Avec ce chargement, l'intégrale dans l'espace de la cellule devient :

$$\begin{aligned}&\frac{1}{s_{cell}}\int \partial_{y_j}\sigma^1_{ij} + \frac{1}{s_{cell}}\int \partial_{Y_j}\sigma^0_{ij} + \frac{1}{s_{cell}}\int F_i \\ =&0 + \frac{1}{s_{mat}}\int_{mat} \partial_{Y_j}\sigma^0_{ij} + \frac{1}{s_{vide}}\int_{vide} \partial_{Y_j}\sigma^0_{ij} + \frac{1}{s_{mat}}\int_{mat} F_i + \frac{1}{s_{vide}}\int_{vide} F_i\end{aligned} \tag{47}$$

s_{mat} et s_{vide} présentent l'air du matériau et l'air du vide dans une cellule. Lorsque nous mettons les contraintes aux valeurs nulles pour le prolongement dans la partie vide de la cellule, nous obtenons :

$$F_i = -\frac{s_{cell}}{s_{mat}}\partial_{Y_j}\langle\sigma^0_{ij}\rangle$$

Alors le problème du deuxième ordre devient :

$$\begin{cases}\partial_{y_j}\sigma^1_{ij} + (C_{ijmn}A^{\nabla}_{mnkl} - \frac{s_{cell}}{s_{mat}}C^{hom}_{ijkl})K_{lkj} \\ \sigma^1_{ij} = C_{ijkl}(y_p)\varepsilon^1_{kl} \\ \varepsilon^1_{ij} = \frac{1}{2}(u^{\nabla}_{ikl}(y_q)\delta_{jm} + u^{\nabla}_{jkl}\delta_{im})K_{lkm} + \frac{1}{2}\left(\partial_{y_j}u^2_i + \partial_{y_i}u^2_j\right) \\ u^2 \text{ est une fonction périodique de } y_i\end{cases} \tag{48}$$

Nous voyons dans la première des équations (17), nous voyons que la terme $(C_{ijmn}A^{\nabla}_{mnkl} - \frac{s_{cell}}{s_{mat}}C^{hom}_{ijkl})K_{lkj}$ présente les forces volumiques aux différentes points d'intégration. Notons que :

$$B^{\nabla}_{ijkl} = C_{ijmn}A^{\nabla}_{mnkl}$$
$$L_{ijkl} = B_{ijkl} - \frac{s_{cell}}{s_{mat}}C^{hom}_{ijkl}$$

Alors l'équation se simplifie comme

$$\sigma_{ij,j} + L_{ijkl}K_{lkj} = \sigma_{ij,j} + f_i = 0 \tag{49}$$

Pour un modèle soumis d'une déformation plane, f_i peut être écrite sous forme d'une multiplication des matrices :

$$\begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} L_{1111} & L_{1122} & \sqrt{2}L_{1121} & L_{1211} & L_{1222} & \sqrt{2}L_{1221} \\ L_{2111} & L_{2122} & \sqrt{2}L_{2121} & L_{2211} & L_{2222} & \sqrt{2}L_{2221} \end{pmatrix} [K] \tag{50}$$

Comme la symétrie de la terme C^{hom}, la terme $L_{\alpha\beta\gamma\delta}$ possède la symétrie entre l'indice de α et β et aussi γ et δ. Par example, L_{1211} (le quatrième coefficient dans la première ligne) égale à L_{2111} (le premier coefficient dans la deuxième ligne de la matrice).

De plus certains termes sont délicates à calculer en distinguant la différence entre la notation ingénieur et la notation de voigt. Par exemple, pour la terme $\sqrt{2}L_{1121}$, on a

$$\sqrt{2}L_{1121} = \sqrt{2}(B^{\nabla}_{1121} - \frac{s_{cell}}{s_{mat}}C^{hom}_{1121}) = \sqrt{2}B^{\nabla}_{1121}$$

mais pour la terme $\sqrt{2}L_{1212}$

$$\sqrt{2}L_{1212} = \sqrt{2}(B^{\nabla}_{1212} - \frac{s_{cell}}{s_{mat}}C^{hom}_{1212}) = \sqrt{2}B^{\nabla}_{1212} - \frac{s_{cell}}{s_{mat}}\frac{C^{hom}_{33}}{\sqrt{2}}$$

dans laquelle C^{hom}_{33} présente la terme $2C^{hom}_{1212}$ dans la matrice de C^{hom}. Nous revenons à la terme B_{1212} avec la prudence. B_{1212} présente la contrainte dans les différents points d'intégration $\sigma_{12}^{E_{12}/\sqrt{2}}$, alors

$$\sqrt{2}L_{1212} = \sqrt{2}\sigma_{12}^{E_{12}} - \frac{s_{cell}}{s_{mat}}\frac{C^{hom}_{33}}{\sqrt{2}} = \sigma_{12}^{\sqrt{2}E_{12}} - \frac{s_{cell}}{s_{mat}}\frac{C^{hom}_{33}}{\sqrt{2}}$$

Pour appliquer les forces volumiques, nous utilisons la subroutine appelée *DLOAD*, la subroutine nous permet d'imposer des forces volumiques aux points d'intégrations dans les différents Steps. Par exemple, pour le Step-K_{111}, nous imposons les forces volumiques L_{1111} dans la direction 1 et L_{2111} dans la direction 2 aux points de Gauss, dans lesquelles L_{1111} est la contrainte σ_{11} dans le Step-E11, et L_{2111} est σ_{12} dans le Step-E_{11}. Donc la matrice de L_{ijkl} est écrite sous cette forme :

$$\begin{pmatrix} L_{1111} & L_{1122} & \sqrt{2}L_{1121} & L_{1211} & L_{1222} & \sqrt{2}L_{1221} \\ L_{2111} & L_{2122} & \sqrt{2}L_{2121} & L_{2211} & L_{2222} & \sqrt{2}L_{2221} \end{pmatrix}$$
$$= [B^{\nabla}](y_q) - \frac{s_{cell}}{s_{mat}} \begin{pmatrix} C^{hom}_{1111} & C^{hom}_{1122} & 0 & 0 & 0 & \frac{C^{hom}_{33}}{\sqrt{2}} \\ 0 & 0 & \frac{C^{hom}_{33}}{\sqrt{2}} & C^{hom}_{1122} & C^{hom}_{2222} & 0 \end{pmatrix}$$

Afin d'implémenter les forces volumiques qu'on a calculées, nous devons les définir à l'aide de code python pendant la création du travail :

```
m = mdb.models[modelname]
m.BodyForce(comp1=1.0, comp2=1.0,comp3=1.0, createStepName='Stepname',distributionType=USER_DEFINED, field='',name = '
    BodyForce',region=y.sets['Cellule'])
```

Sigini

Avant de calculer les précontraintes et les définir à l'aide des subroutines dans Abaqus, nous devons faire une interpolation aux points d'intégrations grâce à la fonction de forme pour l'élément pour $[u^{\nabla}]$, puisque tous les calculs dans les équation (33) et (34) sont faits discrètement aux noeuds en notant que les valeurs du déplacement obtenus du fichier .odb sont aux noeuds. Par exemple, pour un élément de la forme CPE4 qui signifie une fonction de forme linéaire et dans chaque élément, dont la connectivité est quatre et le nombre de point d'intégration est aussi quatre. Voici les numérotations au défaut pour les noeuds et les points d'intégrations dans un élément donné présenté dans le figure 3. Dans notre calcul, nous utilisons l'élément de genre CPE4, qui signifie un élément contient 4 noeuds et 4 points d'intégration, alors la fonction d'interpolation pour un élément unitaire existe à l'aide de la fonction de forme :

$$\begin{aligned} X(r,s) =& \frac{1}{4}(1-r)(1-s)X_1 + \frac{1}{4}(1+r)(1-s)X_2 + \\ & \frac{1}{4}(1+r)(1+s)X_3 + \frac{1}{4}(1-r)(1+s)X_4 \end{aligned} \tag{51}$$

dans laquelle X_i présent les valeurs aux noeuds, $X(r,s)$ présente une valeur interpolée et r et s correspondent aux coordonnées d'un point dans l'élément. Pour le cas d'élément CPE4, la numérotation des points d'intégration et leurs coordonnées associées sont dans le tableau suivant : En utilisant l'équation 51, nous pouvons obtenir les valeurs de $[u^{\nabla}]$ aux points

Tableau 6 – Numérotation des points d'intégration pour l'élément *CPE4* et leurs coordonnées

Numérotation	r (coordonnée sur l'axe x)	s (coordonnée sur l'axe y)
1	$-\sqrt{3}/3$	$-\sqrt{3}/3$
2	$\sqrt{3}/3$	$-\sqrt{3}/3$
3	$-\sqrt{3}/3$	$\sqrt{3}/3$
4	$\sqrt{3}/3$	$\sqrt{3}/3$

d'intégration et les stocker dans une matrice $(N^{ip} \times 2 \times 3)$ dans laquelle N^{ip} qui présente le nombre de point d'intégration dans le modèle.

Pendant l'étape de la validation de code, nous trouvons qu'Abaqus ne lit pas des précontraintes aux points d'intégration dans un élément. Par contre, la valeur moyenne des précontraintes dans les quatre points d'intégration est prise en compte. Alors différentes valeurs de la précontrainte aux points d'intégration vont générer une oscillation des contraintes dans le résultat de la simulation. Donc au lieu d'utiliser la fonction de forme (l'équation (51)), nous mettons la valeur moyenne des quatre nœuds pour les quatre points d'intégration dans cet élément :

$$X(r,s) = \frac{1}{4}X_1 + \frac{1}{4}X_2 + \frac{1}{4}X_3 + \frac{1}{4}X_4$$

Dans ce cas, les précontraintes dans les quatre points d'intégrations sont identiques, alors il n'y aura plus d'oscillation des valeurs de la contrainte dans le résultat.

A partir des équations dans 17, nous voyons que les pré-déplacements sont en fonction de $u^{\nabla}_{ijk}(y)$

$$\varepsilon^{pre}_{ij} = \frac{1}{2}(u^{\nabla}_{ikl}(y_q)\delta_{jm} + u^{\nabla}_{jkl}\delta_{im})E_{lk,m} \tag{52}$$

et les précontraintes associées

$$\sigma^{pre}_{ij} = C_{ijkl}\varepsilon^{pre}_{kl} \tag{53}$$

Pour implanter nos calculs de pré-traitement à l'aide de Python, nous écrivons premièrement l'équation 52 sous forme matricielle :

$$\begin{bmatrix} \varepsilon_{11}^{pre} \\ \varepsilon_{22}^{pre} \\ 2\varepsilon_{12}^{pre} \end{bmatrix} = [u^{\nabla}][K] \tag{54}$$

et l'équation 53 aussi sous la forme matricielle en fonction de la ϵ_{ij}^{pre} selon la notation ingénieur :

$$\begin{pmatrix} \sigma_{11}^{pre} \\ \sigma_{22}^{pre} \\ \sigma_{12}^{pre} \end{pmatrix} = \begin{pmatrix} C_{1111} & C_{1122} & 0 \\ C_{2211} & C_{2222} & 0 \\ 0 & 0 & C_{1212} \end{pmatrix} \begin{pmatrix} \varepsilon_{11}^{pre} \\ \varepsilon_{22}^{pre} \\ 2\varepsilon_{12}^{pre} \end{pmatrix} \tag{55}$$

A l'aide des équation 54 et 55 et les coefficients de $[u^{\nabla}]$ aux points d'intégration, nous calculons les précontraintes aux points d'intégration et les stocker dans les fichiers .unf. Pour implanter ces précontraintes, la subroutine *SIGINI* est utilisée qui nous permet de définir les précontraintes selon la numérotation des éléments et les quatre points d'intégration dedans. Pour caractériser ce fonctionnement d'Abaqus, nous ajoutons une ligne en python pendant la création des travail :

```
m = mdb.models[modelname]
m.Stress(name = 'prestress', distributionType = USER_DEFINED, region = t.sets['Cellule'])
```

dans laquelle il va créer un champ de contrainte dans l'étape initiale et USER_DEFINED signifie qu'Abaqus va chercher automatiquement les précontraintes qui sont définies dans les fichiers de subroutine s'ils existent et "Cellule" représente le domaine du quart de la cellule qu'on veut appliquer la précontrainte.

Dans la pratique, nous trouvons qu'Abaqus ne cherche pas automatiquement la valeur de σ_{33}^{pre} grâce aux propriétés de la déformation plane, donc nous devons le calculer avec une manière un peu artisanale et l'implémenter. Dans le cas de la déformation plane, nous avons

$$0 = \varepsilon_{33} = S_{33\alpha\beta}\sigma_{\alpha\beta} + S_{\alpha\beta 33}\sigma_{33}$$

alors

$$\sigma_{33} = -\frac{S_{33\alpha\beta}}{S_{3333}}\sigma_{\alpha\beta} = \frac{-\frac{\nu}{E}\delta_{\alpha\beta}}{\frac{1}{E}}\sigma_{\alpha\beta} = \nu(\sigma_{11} + \sigma_{22})$$

Donc σ_{33} n'est pas indépendant et possède une relation linéaire en fonction de σ_{11} et σ_{22}.

Validation

Pour vérifier nos formes matricielles et les manière d'usage des subroutines, on lance premièrement le calcul dans un modèle stratifié et comparons les résultats du calcul dans le post-traitement de code python avec le résultat analytique et nous retrouvons les mêmes coefficients du $[C^{hom}]$, $[D]$, $[F]$, $[G]$ et $[H]$ dans une incertitude acceptable.

4 Application à une microstructure pantographique

4.1 Présentation de la cellule

Selon le projet du stage M2 de David Cohen [4], la cellule pantographique a des effets de second gradient. Elle est ainsi conçue pour obéir à un "effet pantographe", c'est-à-dire qu'elle possède une direction ou le déploiement doit se faire librement, sans modification des longueurs

et donc sans nécessiter d'énergie et le coefficient d'élasticité dans cette direction est vers nul. Elle possède donc un mode mou de déploiement et doit logiquement présenter des effets de second gradient. En comportant des zones de vide, le contraste matériel est fixé infini. Un milieu périodique qui est composée d'une microstructure pantographique est présenté dans la figure 4 et sa cellule est encadré par un rectangle rouge. La cellule est définie par les différents

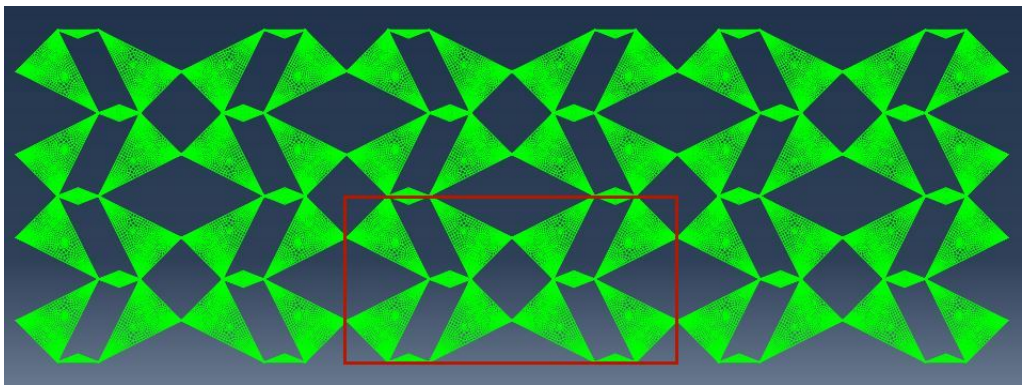

FIGURE 4 – Milieu périodique et sa cellule

paramètres :c, b ($c = 2b$), k, p et les nombreux rayons de courbure qui sera expliqué. La géométrie du quart de la cellule est présentée dans la figure 5. Et nous remarquons que le coefficient de Poisons lors du déploiement est nulle et c'est une condition souhaitable pour l'effet pantographe voulu.

Si on fait l'agrandissement des connections (l'agrandissement de partie verte dans la figure 5) entre les triangles dans une figure détaillé (la figure 6), nous observons plusieurs rayons de courbures significatives. Le rayon du cercle inscrit est souligné par une flèche.

Dans la suite du travail, nous voudrions étudier l'influence de ces rayons de courbures. Pour ces rayons de courbure, nous allons varier les rapports trigonométriques pour les faire dépendre du rayon r d'un seul cercle inscrit virtuel auquel ils sont tous tangents. Ce rayon est alors l'unique paramètre déterminant la taille des connexions internes. Les fonctions des rayons de courbures autour des connections entre les triangles en fonction du rayon de cercle inscrit sont présentées dans la code python de la partie "Paramètres et partition internes de la pantographe" dans les annexes. Nous étudierons l'influence de ce paramètre r dans le problème du premier ordre et deuxième ordre. Les courbures différentes sont aussi présentées dans la figure 13.

Dans les simulations numériques suivantes, nous fixons la taille de la cellule : $l_1 = 4m$, $l_2 = 2m$, $p = 0.2m$, $k = 1.0m$, $c = 2.0m$ et $b = 1.0m$ pour la géométrie du quart de la cellule, présentée dans la figure 5.

4.2 Résultat du problème du premier ordre

Dans cette partie, nous étudions l'effet pantographe et les tendances des coefficients de la matrice $[C^{hom}]$ par rapport aux rayons du cercle inscrit.

Pour le Step-E_{11}, la cellule est soumise à une traction dans la direction 1 et la résistance de la déformation se remarque sur la figure 7. Selon le résultat de la simulation dans Abaqus, la direction 1 n'est pas la direction de mode mou, les déformations de flexions des triangles sont visibles en profondeur dans les solides.

Pour le Step-E_{22}, les contraintes sont présentées dans la figure 8. Dans cette étape, la cellule est soumise à une déformation moyenne dans la direction 2. Pour la visualisation, nous mettons la même valeur maximale de la contrainte dans l'interface Abaqus. En comparant

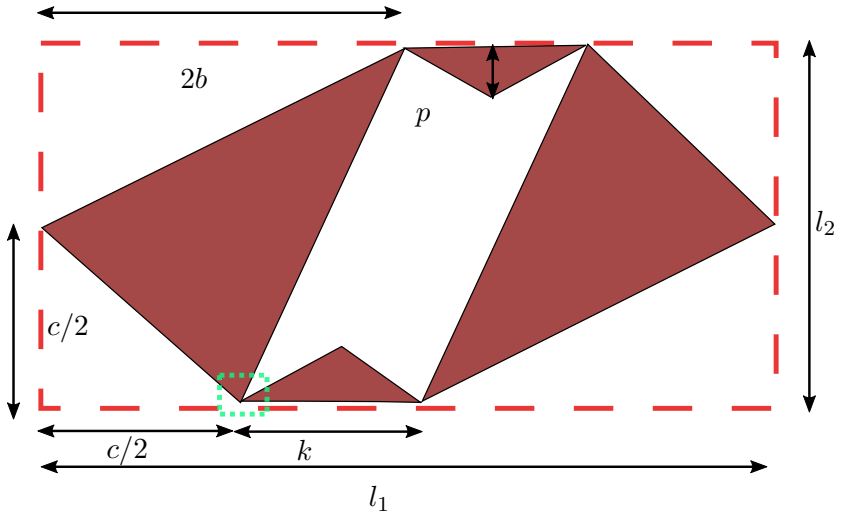

FIGURE 5 – Quart de cellule et sa géométrie

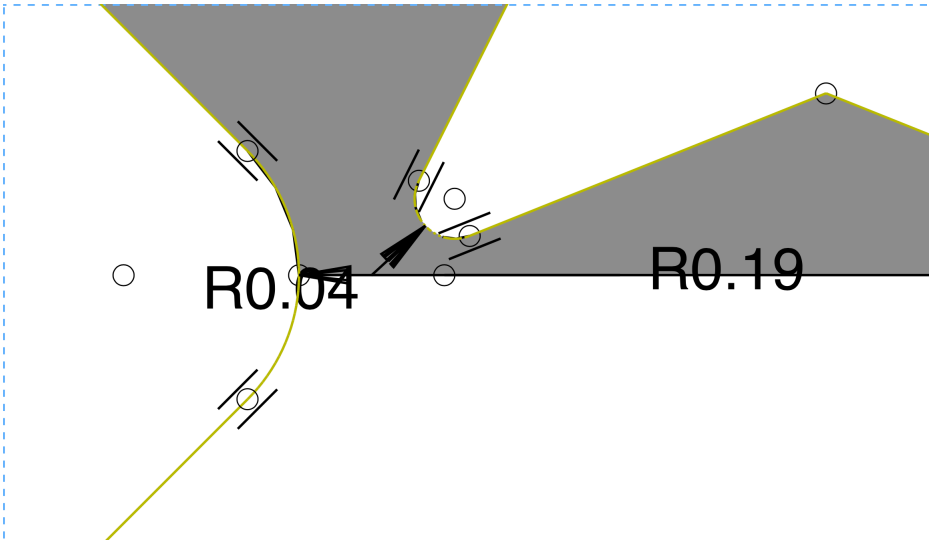

FIGURE 6 – Agrandissement des connections entre les triangles dans une cellule

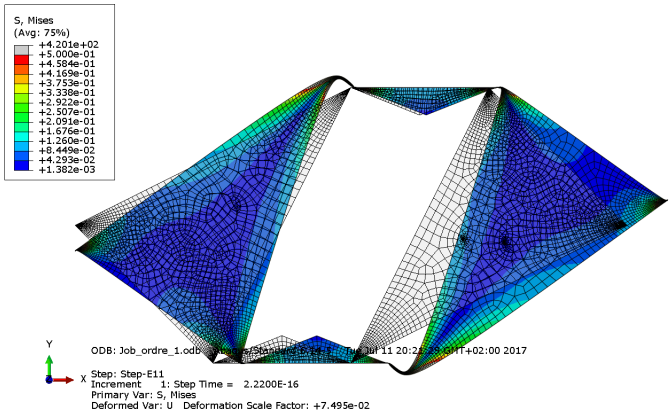

FIGURE 7 – Contraintes dans le Step-E_{11}

avec le résultat du Step-E_{111}, les solides dans la cellule tournent sans réellement se déformer et les déformations restent relativement locales. Alors on peut dire que le pantographe peut se déployer librement dans cette direction et c'est un mode mou.

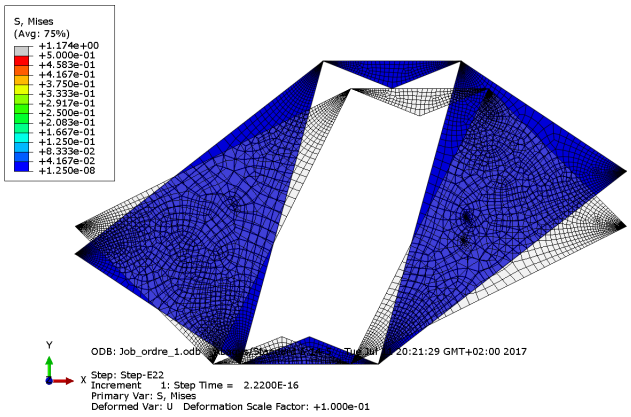

FIGURE 8 – Contraintes dans le Step-E_{22}

L'effet de cisaillement libre dû au mode mou du pantographe est également très visible dans la figure 9.

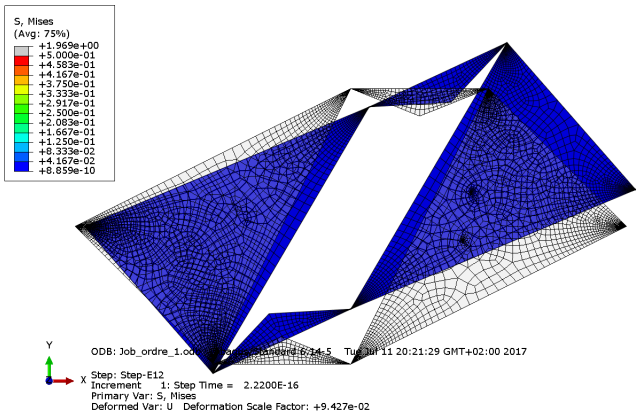

FIGURE 9 – Contraintes dans le Step-E_{12}

Afin d'étudier les tendances des coefficients de la matrice $[C^{hom}]$ par rapport aux rayons du cercle inscrit, nous faisons la même simulation des trois étapes pour les rayons qui tendent vers la limite zéro : r = 0.2, 0.1, 0.07, 0.05, 0.03, 0.01, 0.007, 0.005, 0.003, 0.001, 0.0007, 0.0005, 0.0003, 0.0001 m.

En diminuant le rayon, on réussit à modéliser de mieux en mieux l'effet pantographe. Les tendances des coefficients dans la matrice $[C^{hom}]$ sont présentées dans la figure 10. On remarque que la résistance dans la direction 1 reste dominante par rapport aux autres coefficients même si on diminue le rayon du cercle inscrit et le coefficient d'élasticité dans cette direction (C^{hom}_{1111}) reste toujours apparent et décroit beaucoup plus lentement que les autres coefficients à mesure que l'on tend vers la limite.

Tandis que l'on constate le coefficients C^{hom}_{2222} tend vers zéros quand le rayon de cercle inscrit est proche de zéro, qui nous raconte l'absence d'énergie sollicité dans la direction 2 et une liberté de compressibilité dans cette direction.

La liberté en cisaillement est évident en vue de la décroissance vers zéro du coefficient $C_{1212^{hom}}$. On constate aussi C^{hom}_{1122} décroit aussi très vite en fonction du rayon diminué, mais

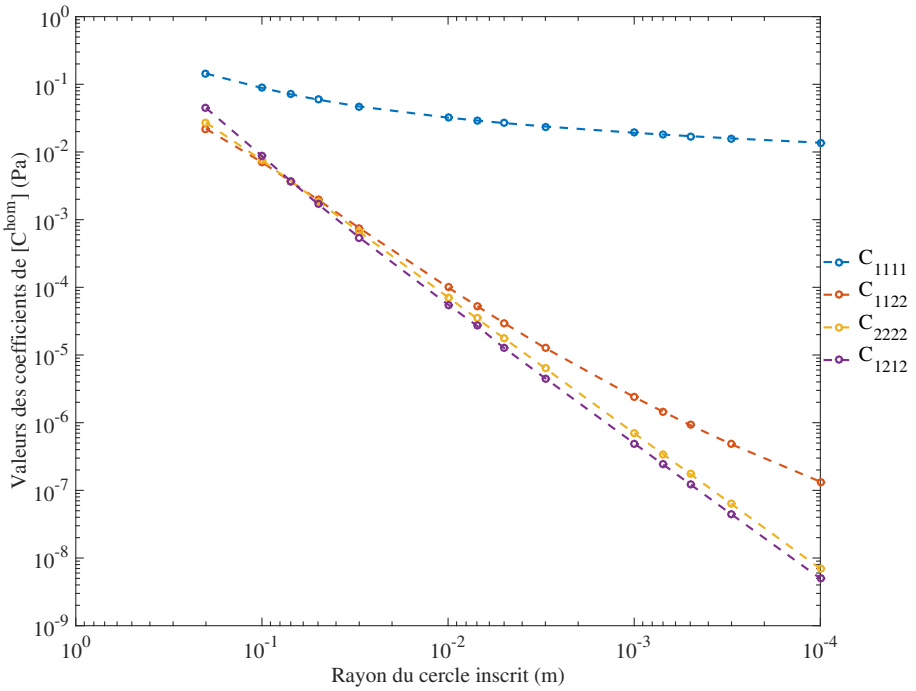

FIGURE 10 – L'évolution des coefficients de $[C^{hom}]$ en fonction du rayon du cercle inscrit

avec une vitesse moins élevée que C^{hom}_{2222} et C^{hom}_{1212}.

Selon la figure 10, quand le rayon du cercle tend vers zéro, nous disons asymptotiquement que C_{2222} et C_{1212} peuvent être écrits avec une fonction polynomiale :

$$C^{hom}_{2222} = C_1 \times r^2 \quad C^{hom}_{1212} = C_2 \times r^2 \tag{56}$$

Le raideur de Cauchy n'étant plus inversible avec ses composantes qui tendent vers zéro quand le rayon de cercle inscrit est proche de la limite, on peut alors rechercher des effets de second gradient.

4.3 Résultat du problème du deuxième ordre

Nous faisons le calcul des raideurs de second gradient dans toutes les directions, c'est-à-dire calculer les coefficients de la matrice $[F]-[G]-[G]^t$. Dans les paragraphes suivants, nous notons $[J] = [F] - [G] - [G]^t$. Les tendances des raideurs de second gradient (les coefficients diagonaux de la matrice $[J]$) sont présentées dans la figure 11. Certains coefficients négatifs de J_{33} et J_{44} sont cachés lorsqu'on fait une figure de l'échelle logarithmique. On propose quand même les déformations des six étapes de l'implémentation Abaqus dans les figures 14, 15, 16, 17, 18 et 19 sur les annexes.

lorsque J_{11} et J_{44} sont liés aux K_{111} et K_{112} (les gradients des déformations macroscopiques dans la direction 1 : E_{11}), respectivement, la longueur caractéristique pour le second gradient K_{111} et K_{112} sont :

$$l_{11} = \sqrt{\frac{J_{11}}{C^{hom}_{1111}}} \qquad l_{44} = \sqrt{\frac{J_{44}}{C^{hom}_{1111}}} \tag{57}$$

de la même raison on obtient :

$$l_{22} = \sqrt{\frac{J_{22}}{C^{hom}_{2222}}} \qquad l_{55} = \sqrt{\frac{J_{55}}{C^{hom}_{2222}}} \tag{58}$$

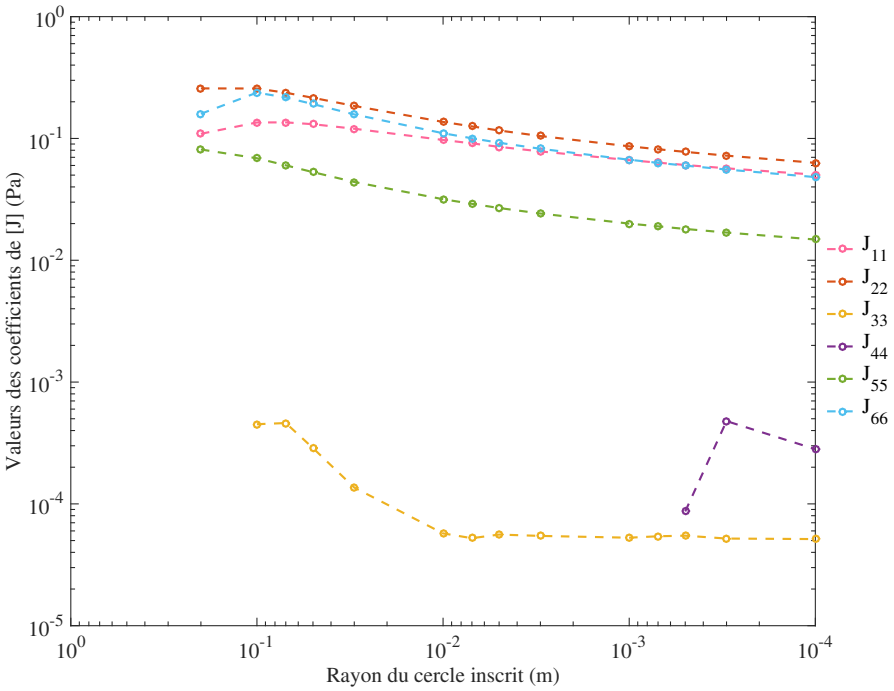

FIGURE 11 – L'évolution des coefficients de $[J]$ en fonction du rayon du cercle inscrit

$$l_{33} = \sqrt{\frac{J_{33}}{C^{hom}_{1212}}} \qquad l_{66} = \sqrt{\frac{J_{66}}{C^{hom}_{1212}}} \tag{59}$$

A l'aide des équations (57), (58) et (59), nous calculons les longueurs caractéristiques des effets de second gradient pour un rayon à la limite. Les longueurs caractéristiques en fonction du rayon inscrit sont présentées dans la figure 12 à l'échelle logarithmique.

Longueur l_{55}

l_{55} est la longueur caractéristique des effets de second gradient dans le déploiement dans la direction 2. On remarque dans la figure 12 que cette longueur caractéristique est bien supérieure à taille d'une cellule ($l_1 = 4m$, $l_2 = 2m$) quand le rayon tend vers zéro. Par exemple, $l_{55}(r = 10^{-4}) \approx 1.45 \times 10^3$, alors des effets de second gradient se propagent sur environs $\dfrac{l_{55}}{2l_2} \approx 363$ cellules de ce pantographe. On remarque aussi sur la figure 11 que le coefficient J_{55} augmente à mesure que l'on se rapproche de la limite et tend à stagner vers asymptote horizontale. Donc l'effet de second gradient est bien détecté dans cette direction de déploiement.

Longueurs l_{11} et l_{44}

Pour le second gradient K_{111}, on peut aussi remarquer qu'en tendant vers la limite, le raideur de second gradient J_{11} tend à stagner vers une limite horizontale et la valeur de la longueur caractéristique tends vers $2m$ quand le rayon inscrit tend vers zéro, qui est quand même inférieur à la longueur horizontale de la cellule $2l_1 = 8m$. Donc il n'y pas d'effet de second gradient pour le déploiement dans la direction 1.

En effet, les longueurs caractéristiques l_{11} et l_{44} sont liés au coefficient C^{hom}_{1111}, alors il n'y

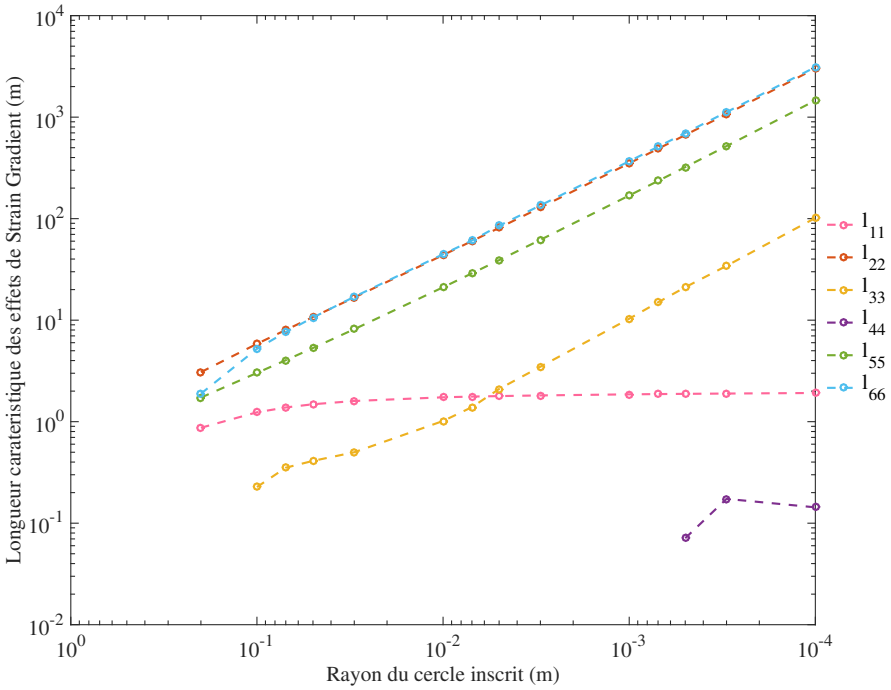

FIGURE 12 – L'évolution des longueurs caractéristiques des effets de second gradient en fonction du rayon du cercle inscrit

pas d'effet de second gradient pour les cas et nous voyons un phénomène curieux qu'il y a certaines valeurs négatives de J_{44}. En effet, on ne sait pas si le résultat est correct puisqu'on applique le nombre de maillage limité autour des connections entre les triangles et il peut tendre vers 0 sans qu'on le voie.

Longueurs l_{22} et l_{66}

Sur la figure 12, on peut aussi remarquer, d'une manière globale, qu'en tendant vers la limite, le raideur de second gradient J_{22} et J_{66} tend à stagner vers une limite horizontale. Or C^{hom}_{1111} tends vers une limite et C^{hom}_{2222} et C^{hom}_{1212} décroit selon une fonction asymptotique dans la fonction (56), selon la figure 10. Quant à la limite du rayon, l_{22} et l_{66} sont proportionnels au $1/r$.

A l'aide de l'équation (25), on remarque l_{22} est la longueur caractéristique des effets de second gradient K_{221} et l_{66} est celle de K_{122}. Le second gradient K_{221} signifie une variation dans la direction 1 de la déformation dans la direction 2. Le second gradient K_{122} signifie une variation dans la direction 2 du cisaillement, c'est-à-dire une flexion des gradient des cisaillements. On a vu que l_{22} et l_{66} augmente proportionnellement au $1/r$, alors cela nous raconte que ces deux phénomènes se propagent sur beaucoup de cellules pour un rayon proche de la limite, comme le propagation d'effet de second gradient sur le déploiement dans la direction 2.

Longueur l_{33}

L'effet de second gradients K_{121} est vraiment étrange, car ils ont à la fois les termes négatifs. En effet, pour l'homogénéisation du deuxième ordre, J_{33} n'a pas la peine d'être toujours positifs. Grâce à l'équation (25), nous voyons que l'énergie est composée des trois

parties : l'énergie des déformations macroscopiques, des second gradients et les deuxièmes gradients des déformations, alors les coefficients non principaux n'ont pas la peine d'être toujours positifs pour assurer l'existence de la solution correspondent à la minimisation de l'énergie dans l'équation (25).

5 Conclusion

Dans cet article, nous avons appliqué une homogénéisation d'ordre supérieur en utilisant un développement asymptotique du déplacement dans un cas où le tenseur d'élasticité homogénéisé dans un milieu périodique est singulier, c'est-à-dire certains coefficients sont nuls. Nous commençons à établir les propriétés macroscopiques en fonction de la déformation macroscopique et le second gradient du déplacement et écrire l'énergie de déformation avec plusieurs ordres supérieur, qui prévient un effet de second gradient et une longueur caractéristique. Dans la deuxième partie, nous adoptons une notations de Voigt pour les formes matricielles du tenseurs et considérons l'effet des symétries dans un quart de cellule, afin de implémenter dans Abaqus, un logiciel de la simulation numérique. Les subroutines des forces volumiques (*DLOAD*) et des précontraintes (*SIGINI*) sont aussi utilisés dans l'implémentation. Une validation dans un modèle simple qui possède déjà les solutions analytiques sont lancées pour vérifier notre calcul. Dans la dernière partie, nous appliquons dans une microstructure pantographique, dont certain coefficients du tenseur d'élasticité homogénéisé sont nuls et cherchons l'effet de second gradient. Nous vérifions notre intuition et calculons aussi les tendances des certains coefficients des tenseurs. Finalement, nous trouvons les longueurs caractéristiques dans les différentes directions et expliquons leurs tendances.

6 Réflexion

C'est un stage de recherche d'une manière découverte. Au début du stage, j'ai commencé a lire la polycopié du cours de l'homogénéisation de Karam Sab et les descriptions de l'homogénéisation en gradient d'ordre supérieur. Pour de gagner l'expérience des codes de python, j'ai lu les codes implémentées pour l'homogénéisation "Classique" écrites par David Cohen, qui est un étudiant aussi encadré par Arthur Lebée. Pour implémenter les forces volumiques et les précontraintes, j'ai lu les codes de Fortran 77 écrites par Lorenzo Franzoni.

Lors du développement de ce stage il y a eu d'avantages apportés à l'École des Ponts et Chaussées et vice-versa. Le stage chez l'école de Ponts m'a permis entre autre de :

1. Comprendre mieux le principe de l'homogénéisation.
2. Avoir une connaissance du logiciel Abaqus, non seulement dans l'interface CAE, mais aussi des codes pythons interprétés par le CAE kernel
3. Développer les subroutines correspondantes aux problèmes demandés, inclus *DLOAD* pour les forces volumiques et *SIGINI* pour les précontraintes

En parallèle, il existe d'avantages que j'ai eu l'occasion d'apporter à mon tuteur et à l'école des Ponts :

1. Les codes de python pour les créations des travail dans Abaqus et les calculs matriciels dans le post-traitement des données à l'homogénéisation en gradient d'ordre supérieur
2. Les codes de Fortran 77 pour les subroutines en utilisant les adresses dynamiques pour lire les fichiers .unf

Ce stage m'a aussi permis de découvrir la vie de recherche et des discussions quotidiennes sur un projet scientifique avec mon tuteur. Les possibilités des communications avec les autres

doctorats dans le même bureau, des discussions scientifiques m'a permis de développer un esprit de la collaboration. De plus, les cours enseignés par mon tuteur ont été mises en valeur pour l'avancement et bon déroulement du travail.

En outre, l'aperçue d'une véritable culture dans une école de recherche scientifique tel que celle de l'école des Ponts, ont permis d'évoluer en termes de rigueur dans les procédures à suivre aidées par une organisation de la recherche scientifique, qui sera sans doute un avantage pour mon programme d'une mastère spécialisée dans le domaine de la génie mécanique à l'Université de Californie à Berkeley dans le semestre suivant.

En ce qui concerne mon travail, je suis très satisfait du projet qui m'a été proposé. Les différentes tâches que j'ai effectuées durant toute la durée de ce stage m'ont permis de mettre en œuvre les connaissances de l'homogénéisation acquises durant ma formation, de découvrir des domaines qui m'étaient inconnus auparavant, et surtout de me projeter dans mon avenir professionnel.

L'expérience du stage m'a également rassuré la crédité du choix du parcours d'approfondissement pour la quatrième année et aussi l'intérêt de la recherche de la simulation numérique, et m'a rendu plus confiant de poursuivre mes objectifs scientifiques et professionnels.

Bibliographie

[1] ABAQUS Inc. *Abaqus Scripting User's Manual 6.12*. 2012. pdf version.

[2] Karam Sab and Luc Dormieux. *L'homogénéisation des milieux périodiques*. ENPC, 2004. Course book.

[3] V.P Smyshlyaev and K.D. Cherednichenko. On rigorous derivation of strain gradient effects in the overall behavior of periodic heterogeneous media. 2000.

[4] David Cohen. Investigation d'un métamatériau en élasticité. Master's thesis, Laboratoire Navier, ENPC, IFSTTAR, CNRS, 2016.

7 Annexes

7.1 Figures

Géométrie de la cellule

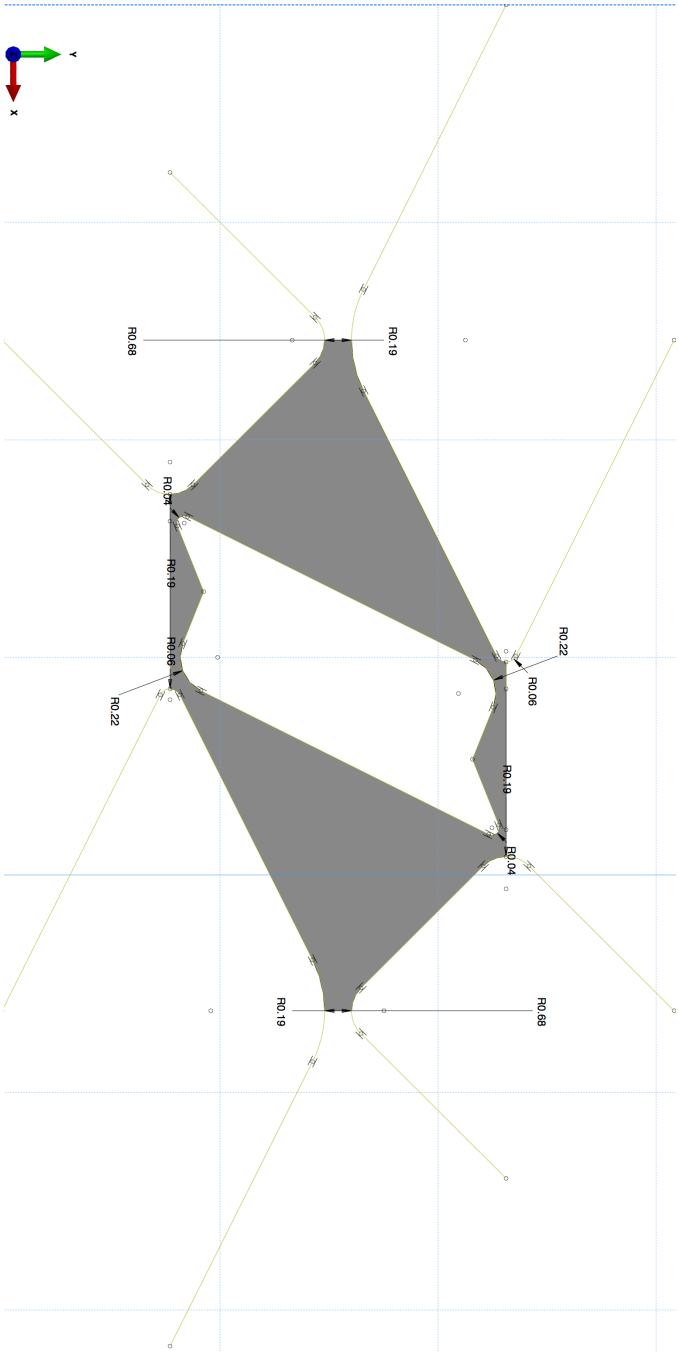

FIGURE 13 – La géométrie d'un quart de la cellule ($r = 0.08$)

Contraintes et déformations symétriques ou antisymétriques dans le problème du deuxième ordre

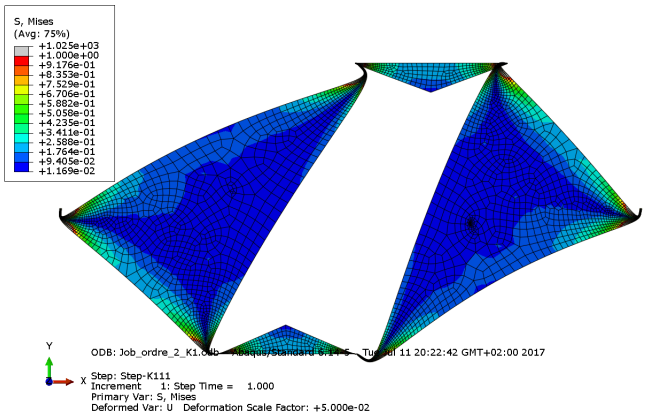

FIGURE 14 – Contraintes dans le Step-K_{111}

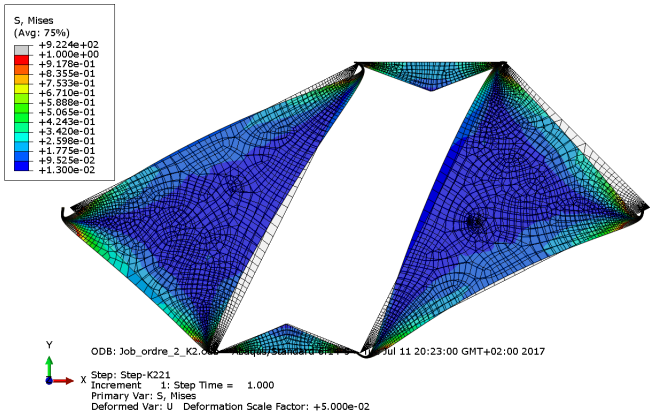

FIGURE 15 – Contraintes dans le Step-K_{221}

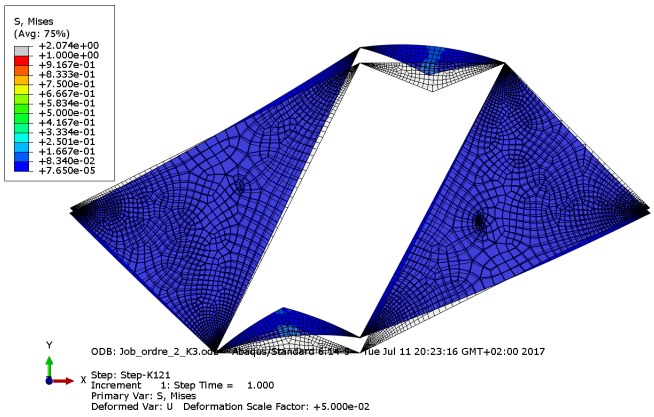

FIGURE 16 – Contraintes dans le Step-K_{121}

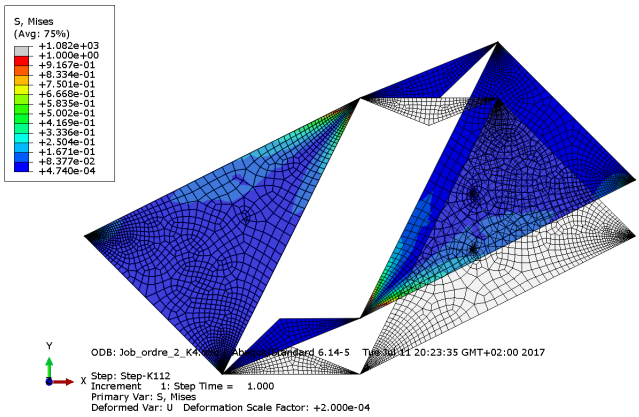

FIGURE 17 – Contraintes dans le Step-K_{112}

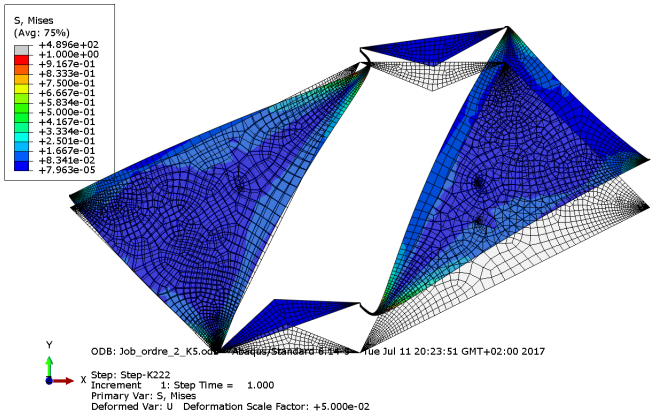

FIGURE 18 – Contraintes dans le Step-K_{222}

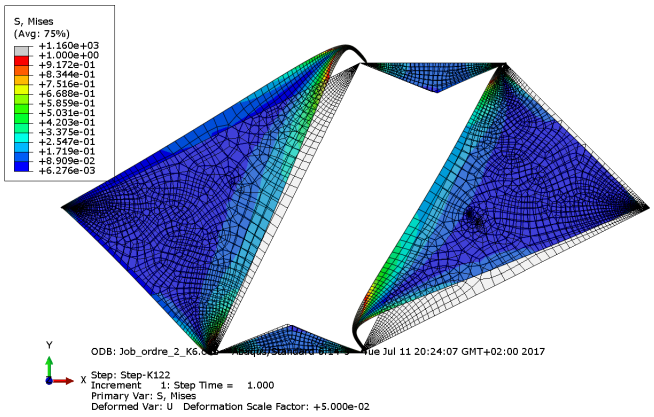

FIGURE 19 – Contraintes dans le Step-K_{122}

7.2 Codes

Python

Processus du calcul de l'homogénéisation

```
import numpy
import timeit
from MyFortranfile import FortranFile
from Matrix_C import *
from symmetry_define import *
global Stiffness
global D_hom
r = 0.001
modelname = 'Pantographe'
execfile(modelname+'.py')
execfile('ordre_1.py')
execfile('Post_Traitement_ordre_1.py')
execfile('C_hom_calculation.py')
execfile('Matrix_C_list.py')
execfile('Precontrainte_Storage.py')
execfile('integrationpoints_force_storage.py')
execfile('D_calculation.py')
execfile('ordre_2.py')
execfile('Post_Traitement_ordre_2.py')
execfile('F_calculation.py')
execfile('G_calculation.py')
execfile('affichage.py')
stop = timeit.default_timer()
print stop - start
```

Paramètres et partition internes de la pantographe

```
# -*- coding: mbcs -*-
from __future__ import division
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from odbAccess import *
import numpy
import mesh
import math

############  parametres geometriques
## attention a la coherence des parametres entre eux
m= mdb.Model(name='Pantographe', modelType=STANDARD_EXPLICIT)
bb =1.0

k=1.0 # longeur connexion
pp=2*bb/10 # epaisseur connexion
c=2.0 # diagonale carre

# r = 0.0003 #0.0005 # 0.0003# rayon du cercle inscrit virtuel dont dependent tous les RdC
# dimensions cellule
l1 = 2*bb+k+(c/2)
l2 = 2*bb # l2 doit etre de meme taille que le 1er terme de l1

# beta = atan (pp/(k/2)) en radian   =   180* ( atan (pp/(k/2))  ) / pi  en degres
# angles pr RDC en radian
alpha1= atan( ((2*bb)-(c/2))/ 2*bb )
alpha2= (pi + alpha1 - atan (pp/(k/2)) - pi*0.5 ) /2
alpha3= (0.5*pi-atan (pp/(k/2))-alpha1)/2
alpha4= pi*45.0/180
alpha5= (pi*90/180) - alpha1

r1=((   sqrt((r*tan(alpha1))**2 + r**2) + r*tan(alpha1)  ) / cos(alpha1)  ) - r
r2=((   sqrt((r*tan(alpha2))**2 + r**2) + r*tan(alpha2)  ) / cos(alpha2)  ) - r
r3=((   sqrt((r*tan(alpha3))**2 + r**2) + r*tan(alpha3)  ) / cos(alpha3)  ) - r
r4=((   sqrt((r*tan(alpha4))**2 + r**2) + r*tan(alpha4)  ) / cos(alpha4)  ) - r
r5=((   sqrt((r*tan(alpha5))**2 + r**2) + r*tan(alpha5)  ) / cos(alpha5)  ) - r

############  parametres materiaux
Ev = 0.001
Young = 1
Poisson = 0.3

############ parametres maillage
n=5.0
minSizeMesh = 2*pi*r/(3*n) # r ? r/n ?
maxSizeMesh = l1/50
maxSizeMesh2 = l1/100  # pour les connexions
forme = QUAD  # forme des elements ( TRI pour triangulaire )
elemcodtyp1 = CPE4  # elements quadratiques DefPlanes  ( CPE8R pr quadra , CPE4R pr lineaire)
elemcodtyp2 = CPE3 # elements quadratiques DefPlanes  ( CPE6M  pr quadra  , CPE3 pr lineraire)
```

```
62
63 ############  parametres post processing
64 Amp = 1.0
65 BCSym = 1
66
67 ############ PART rectangle ################################
68 m.ConstrainedSketch(name='___profile___', sheetSize=200.0)
69 pr = mdb.models['Pantographe'].sketches['___profile___']
70 pr.rectangle(point1=(0.0, l2),point2=(l1, 0.0))
71 m.Part(dimensionality=TWO_D_PLANAR, name='Part-1', type=DEFORMABLE_BODY)
72 m.parts['Part-1'].BaseShell(sketch= pr)
73
74 #### paramterisation des arretes du rectangle
75 p = mdb.models['Pantographe'].parts['Part-1']
76 e = p.edges
77
78 # repere grid point bas gauche
79 edges = e.findAt(((l1/2,l2, 0.0),))
80 p.Set(edges=edges, name='haut')
81
82 edges = e.findAt(((l1,l2/2, 0.0),))
83 p.Set(edges=edges, name='droite')
84
85 edges = e.findAt(((l1/2,0.0, 0.0),))
86 p.Set(edges=edges, name='bas')
87
88 edges = e.findAt(((0.0,l2/2, 0.0),))
89 p.Set(edges=edges, name='gauche')
90
91 ### partition de la geometrie interne
92 m.ConstrainedSketch(gridSpacing=5.19, name='___profile___',
93 sheetSize=207.84, transform= p.MakeSketchTransform(
94     sketchPlane=p.faces[0],
95     sketchPlaneSide=SIDE1, sketchOrientation=RIGHT, origin=(0.0, 0.0, 0.0)))
96 p.projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=m.sketches['___profile___'])
97 mdb.models['Pantographe'].sketches['___profile___'].sketchOptions.setValues(
98     gridOrigin=(-l1/2, -l2/2))
99 pr = mdb.models['Pantographe'].sketches['___profile___']
100 s = pr.geometry
101
102 ## repere partition toujours point bas gauche contrairement a letoile
103 pr.Line(point1=(0.0,c/2),point2=(c/2,0.0))   #a (A-F)
104 pr.Line(point1=(0.0,-c/2),point2=(c/2,0.0))  #b (A3-F)
105 pr.Line(point1=(0.0,c/2),point2=(-c/2,0.0))  #c (A-F2)
106
107 #* curve a-b  c3
108 pr.FilletByRadius(curve1= s.findAt((  (c/2)/2, (c/2)/2    )),
109 curve2=s.findAt((  (c/2)/2, -(c/2)/2   )),
110 nearPoint1=((c/2)/2, (c/2)/2), nearPoint2=((c/2)/2, -(c/2)/2), radius=r4)
111
112 #* curve c-a  c10
113 pr.FilletByRadius(curve1= s.findAt((  (c/2)/2, (c/2)/2   )),
114 curve2=s.findAt((  -(c/2)/2, (c/2)/2   )),
115 nearPoint1=((c/2)/2, (c/2)/2), nearPoint2=(-(c/2)/2, (c/2)/2), radius=r4)
116
117 pr.Line(point1=((2*bb)+k+(c/2),2*bb-(c/2) ) , point2=(   (2*bb)+k,2*bb   ))  #d (D-C)
118 pr.Line(point1=((2*bb)+k+(c/2),2*bb-(c/2) ) , point2=(   (2*bb)+k+c,2*bb ))  #e (D-C2)
119 pr.Line(point1=((2*bb)+k,2*bb) , point2=((2*bb)+k+(c/2),(2*bb)+(c/2) ) ) #f (C-D3)
120
121 #* curve d-e  c9
122 pr.FilletByRadius(curve1= s.findAt((  ((2*bb)+k+(c/2)+(2*bb)+k)/2, (2*bb-(c/2)+2*bb)/2   )),
123 curve2=s.findAt((  ((2*bb)+k+(c/2)+(2*bb)+k+c)/2, (2*bb-(c/2)+2*bb)/2  )),
124 nearPoint1=(( (2*bb)+k+(c/2)+(2*bb)+k)/2, (2*bb-(c/2)+2*bb)/2), nearPoint2=(((2*bb)+k+(c/2)+(2*bb)+k+c)/2, (2*bb-(c/2)+2*bb
        )/2   ), radius=r4)
125
126 #* curve d-f  c4
127 pr.FilletByRadius(curve1= s.findAt((  ( (2*bb)+k+(c/2)+(2*bb)+k)/2, (2*bb-(c/2)+2*bb)/2   )),
128 curve2=s.findAt(( ((2*bb)+k+ (2*bb)+k+(c/2))/2, (2*bb+(2*bb)+(c/2))/2  )),
129 nearPoint1=(( (2*bb)+k+(c/2)+(2*bb)+k)/2, (2*bb-(c/2)+2*bb)/2), nearPoint2=(((2*bb)+k+ (2*bb)+k+(c/2))/2, (2*bb+(2*bb)+(c
        /2))/2 ), radius=r4)
130
131 pr.Line(point1=(k+(c/2),0.0),point2=((2*bb)+k+(c/2),2*bb-(c/2)))  #n (E-D)
132 pr.Line(point1=(2*bb+k+(c/2),(2*bb)-(c/2)),point2=((c/2)+k+(4*bb),0.0))  #o (D-E2)
133 pr.Line(point1=(0.0,c/2),point2=(2*bb,2*bb))  #p (A-B)
134 pr.Line(point1=(0.0,c/2),point2=(-(2*bb),2*bb))  #q (A-B2)
135 pr.Line(point1=(k+(c/2),0.0),point2=((2*bb)+k+(c/2),-(2*bb)+(c/2)))  #n (E-D2)
136 pr.Line(point1=(2*bb,2*bb),point2=(0.0,(4*bb)-(c/2)))  #r (B-A2)
137
138 # curve m-o  c11
139 pr.FilletByRadius(curve1= s.findAt((  (k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+2*bb-(c/2))/2   )),
140 curve2=s.findAt((  (2*bb+k+(c/2)+(c/2)+k+(4*bb))/2, ((2*bb)-(c/2)+0.0)/2  )),
141 nearPoint1=((k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+2*bb-(c/2))/2), nearPoint2=((2*bb+k+(c/2)+(c/2)+k+(4*bb))/2, ((2*bb)-(c/2)
        +0.0)/2  ), radius=r5)
142
143 # curve q-p  c12
144 pr.FilletByRadius(curve1= s.findAt((  (0.0+2*bb)/2, ((c/2)+2*bb)/2   )),
145 curve2=s.findAt((  (0.0-(2*bb))/2, (c/2+2*bb)/2  )),
146 nearPoint1=((0.0+2*bb)/2, ((c/2)+2*bb)/2), nearPoint2=((0.0-(2*bb))/2, (c/2+2*bb)/2  ), radius=r5)
147
148 # curve m-n c2
149 pr.FilletByRadius(curve1= s.findAt((  (k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+-(2*bb)+(c/2))/2   )),
150 curve2=s.findAt((     (k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+2*bb-(c/2))/2     )),
151 nearPoint1=(     (k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+-(2*bb)+(c/2))/2     )  , nearPoint2=(    (k+(c/2)+(2*bb)+k+(c/2))/2,
        (0.0+2*bb-(c/2))/2     )  , radius=r1)
152
153 # curve p-r  c1
```

```
pr.FilletByRadius(curve1= s.findAt((  (0.0+2*bb)/2, (c/2+2*bb)/2   )),
curve2=s.findAt((     (2*bb+0.0)/2, (2*bb+(4*bb)-(c/2))/2     )),
nearPoint1=(     (0.0+2*bb)/2, (c/2+2*bb)/2      )  , nearPoint2=(  (2*bb+0.0)/2, (2*bb+(4*bb)-(c/2))/2     ) , radius=r1)

pr.Line(point1=(c/2,0.0),point2=(2*bb,2*bb))  #g (F-B)
pr.Line(point1=(c/2,0.0),point2=((c/2)+(k/2),pp))  #i (F-G)
pr.Line(point1=(k+(c/2),0.0),point2=((2*bb)+k,2*bb))  #h (E-C)
pr.Line(point1=((2*bb)+(k/2),(2*bb)-pp),point2=((2*bb)+k,2*bb))  #l (H-C)
pr.Line(point1=((c/2)+(k/2),pp),point2=(k+(c/2),0.0))  #j (G-E)
pr.Line(point1=(2*bb,2*bb),point2=((2*bb)+(k/2),(2*bb)-pp))  #k (B-H)

#curve g-i  c7
pr.FilletByRadius(curve1= s.findAt((  (c/2+2*bb)/2, (0.0+2*bb)/2   )),
curve2=s.findAt((  (c/2+(c/2)+(k/2))/2, (0.0+pp)/2  )),
nearPoint1=(  (c/2+2*bb)/2, (0.0+2*bb)/2 ) , nearPoint2=(     (c/2+(c/2)+(k/2))/2, (0.0+pp)/2  )  , radius=r3)

###curve h-l  c6
pr.FilletByRadius(curve1= s.findAt((  (k+(c/2)+(2*bb)+k)/2, (0.0+2*bb)/2   )),
curve2=s.findAt((     ((2*bb)+(k/2)+(2*bb)+k)/2, ((2*bb)-pp+2*bb)/2     )),
nearPoint1=(     (k+(c/2)+(2*bb)+k)/2, (0.0+2*bb)/2      )  , nearPoint2=(    ((2*bb)+(k/2)+(2*bb)+k)/2, ((2*bb)-pp+2*bb)/2
      ) , radius=r3)

###curve g-k c5
pr.FilletByRadius(curve1= s.findAt((  (c/2+2*bb)/2, (0.0+2*bb)/2   )),
curve2=s.findAt((     (2*bb+(2*bb)+(k/2))/2, (2*bb+(2*bb)-pp)/2     )),
nearPoint1=(     (c/2+2*bb)/2, (0.0+2*bb)/2      )  , nearPoint2=(     (2*bb+(2*bb)+(k/2))/2, (2*bb+(2*bb)-pp)/2      ) , radius
      =r2)

###curve j-h c8
pr.FilletByRadius(curve1= s.findAt((  ((c/2)+(k/2)+k+(c/2))/2, (pp+0.0)/2   )),
curve2=s.findAt((     (k+(c/2)+(2*bb)+k)/2, (0.0+2*bb)/2     )),
nearPoint1=(     ((c/2)+(k/2)+k+(c/2))/2, (pp+0.0)/2      )  , nearPoint2=(    (k+(c/2)+(2*bb)+k)/2, (0.0+2*bb)/2     ) ,
      radius=r2)

### partitions des diamants
pr.Line(point1=(c/2-r,0.0),point2=(c/2+r,0.0))
pr.Line(point1=(c/2+k-r,0.0),point2=(c/2+k+r,0.0))
pr.Line(point1=(2*bb-r,2*bb),point2=(2*bb+r,2*bb))
pr.Line(point1=(2*bb+k-r,2*bb),point2=(2*bb+k+r,2*bb))



p = mdb.models['Pantographe'].parts['Part-1']
p.PartitionFaceBySketch(faces=p.faces.findAt(((0.0, 0.0, 0.0), )),sketch=pr)
###### supprimer la partie vide ######
f = p.faces
index_pantographe = mdb.models['Pantographe'].parts['Part-1'].faces.findAt((l1/4,l2/2,0.0),).index
mdb.models['Pantographe'].parts['Part-1'].RemoveFaces(deleteCells=False,
    faceList=f[0:index_pantographe]+f[index_pantographe+1:6])

################## proprietes ############
m = mdb.models['Pantographe']
p = m.parts['Part-1']
m.Material(name='Material-plein')
m.materials['Material-plein'].Elastic(table=((Young, Poisson),))
m.HomogeneousSolidSection(material='Material-plein',
name='plein', thickness=None)

## repere point bas gauche
p.Set(faces=m.parts['Part-1'].faces.findAt(((c/2, bb/4, 0.0), )), name='Cellule')
p.SectionAssignment(offset=0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=p.sets['Cellule'], sectionName='plein',
      thicknessAssignment=FROM_SECTION)

################# Assembly ##############
m.rootAssembly.DatumCsysByDefault(CARTESIAN)
m.rootAssembly.Instance(dependent=ON, name='Part-1-1', part=m.parts['Part-1'])

###################### mesh ##################
# repere Grid ( bas gauche )

#g
pickedEdges = e.findAt(((      (c/2+2*bb)/2, (0.0+2*bb)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)
# i
pickedEdges1 = e.findAt(((      (c/2+(c/2)+(k/2))/2, (0.0+pp)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=SINGLE, end1Edges=pickedEdges1, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)
# j
pickedEdges2 = e.findAt(((      ((c/2)+(k/2)+k+(c/2))/2, (pp+0.0)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=SINGLE, end2Edges=pickedEdges2, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)
# k
pickedEdges2 = e.findAt(((      (2*bb+(2*bb)+(k/2))/2, (2*bb+(2*bb)-pp)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=SINGLE, end1Edges=pickedEdges1, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)
# l
pickedEdges1 = e.findAt(((      ((2*bb)+(k/2)+(2*bb)+k)/2, ((2*bb)-pp+2*bb)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=SINGLE, end2Edges=pickedEdges2, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)
#h
pickedEdges = e.findAt(((      (k+(c/2)+(2*bb)+k)/2, (0.0+2*bb)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)
# p
pickedEdges = e.findAt(((      (0.0+2*bb)/2, ((c/2)+2*bb)/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)
# m
pickedEdges = e.findAt(((      (k+(c/2)+(2*bb)+k+(c/2))/2, (0.0+2*bb-(c/2))/2, 0.0      ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)
# a
pickedEdges = e.findAt(((      (c/2)/2, (c/2)/2, 0.0      ),))
```

```
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)
# d
pickedEdges = e.findAt(((        ((2*bb)+k+(c/2)+(2*bb)+k)/2, (2*bb-(c/2)+2*bb)/2, 0.0          ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh, constraint=FINER)

### arretes connexion x et y
pickedEdges = e.findAt(((        (c/2+(c/2)+k)/2, (0.0+0.0)/2, 0.0          ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)
pickedEdges = e.findAt(((         (2*bb+2*bb+k)/2, (2*bb+2*bb)/2, 0.0           ),))
p.seedEdgeByBias(biasMethod=DOUBLE, endEdges=pickedEdges, minSize=minSizeMesh, maxSize=maxSizeMesh2, constraint=FINER)

#### elements constants sur partitions part et dautre des arrets x y
pickedEdges = e.findAt(((         (c/2-r+c/2+r)/2, (0.0+0.0)/2, 0.0           ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
pickedEdges = e.findAt(((        (c/2+k-r+c/2+k+r)/2, (0.0+0.0)/2, 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
pickedEdges = e.findAt(((         (2*bb-r+2*bb+r)/2, (2*bb+2*bb)/2, 0.0           ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
pickedEdges = e.findAt(((         (2*bb+k-r+2*bb+k+r)/2, (2*bb+2*bb)/2, 0.0           ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)

## elements constants sur les rdc
#c1
pickedEdges = e.findAt(((      2*bb-r, 2*bb-0.00001, 0.0         ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c2
pickedEdges = e.findAt(((         (c/2)+k+r, 0.00001, 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c3
pickedEdges = e.findAt(((         (c/2)-r, 0.00001, 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c4
pickedEdges = e.findAt(((        2*bb+k+r, 2*bb-0.00001, 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
###c5
pickedEdges = e.findAt(((        2*bb+r*cos(atan ( pp/(k/2) )+alpha2),2*bb-r*sin(atan ( pp/(k/2) )+alpha2) , 0.0          )
    ,))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
###c6
pickedEdges = e.findAt(((       2*bb+k-r*cos(atan ( pp/(k/2) )+alpha3)  ,2*bb-r*sin(atan ( pp/(k/2) )+alpha3) , 0.0
    ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
###c7
pickedEdges = e.findAt(((        c/2+r*cos(alpha3+atan ( pp/(k/2) )) ,r*sin(alpha3+atan ( pp/(k/2) )),0.0      ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
###c8
pickedEdges = e.findAt(((       c/2+k-r*cos(alpha2+atan (pp/(k/2)))    ,r*sin(alpha2+atan ( pp/(k/2) )) , 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c9
pickedEdges = e.findAt(((           (2*bb)+k+(c/2)-0.00001, 2*bb-(c/2)+r, 0.0          ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c10
pickedEdges = e.findAt(((           0.00001, (c/2)-r, 0.0                              ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c11
pickedEdges = e.findAt(((          (2*bb)+k+(c/2)-0.00001, 2*bb-(c/2)-r, 0.0           ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)
#c12
pickedEdges = e.findAt(((           0.00001, (c/2)+r, 0.0                              ),))
p.seedEdgeBySize(edges=pickedEdges, size=minSizeMesh, deviationFactor=0.1,constraint=FINER)

###*******
elemType1 = mesh.ElemType(elemCode=elemcodtyp1, elemLibrary=STANDARD) # CPE4R lin / CPE8R quad
secondOrderAccuracy=OFF, hourglassControl=DEFAULT,
distortionControl=DEFAULT)
elemType2 = mesh.ElemType(elemCode=elemcodtyp2, elemLibrary=STANDARD) # CPE3 lin / CPE6M quad
p = mdb.models['Pantographe'].parts['Part-1']
f = p.faces
faces = f.findAt(    (    (c/2, bb/4, 0.0),    )     )
pickedRegions =(faces, )
p.setElementType(regions=pickedRegions, elemTypes=(elemType1, elemType2))
pickedRegions = f.findAt(   ( (c/2, bb/4, 0.0), ))
p.setMeshControls(regions=pickedRegions, elemShape=forme)
p = mdb.models['Pantographe'].parts['Part-1']
p.generateMesh()
cc = mdb.models['Pantographe']
t= cc.rootAssembly.instances['Part-1-1']

# referencepoint = (c/2 - r , 0 , 0)
referencepoint = (0 , 0 , 0)
Matrix1_C = isotrope2d(Young, Poisson)
MaterialSets_C={'Cellule' :Matrix1_C}
Matrix2_C = isotrope2d_nsym(Young, Poisson)
NewMaterialSets_C = {'Cellule' :Matrix2_C}
Matrix_S = isotrope2d_inverse(Young, Poisson)
MaterialSets_S={'Cellule' :Matrix_S}
mdb.models[modelname].rootAssembly.Set(name='Set-1', vertices=t.vertices.findAt(((c/2 - r, 0.0, 0.0), )))
```

Problème du premier ordre et ses conditions aux limites

```
################ 4 STEP  ##############
m.StaticLinearPerturbationStep(name='Step-E11',previous= 'Initial')
m.StaticLinearPerturbationStep(name='Step-E22',previous='Step-E11')
m.StaticLinearPerturbationStep(name='Step-E12',previous= 'Step-E22')
m.fieldOutputRequests['F-Output-1'].setValues(variables=('S', 'E', 'U' , 'IVOL','EVOL'))
m.historyOutputRequests['H-Output-1'].setValues(variables=('ALLSE', ))
```

```
#############BCs#################
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E11',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-1', region=t.sets['droite'],
u1= l1, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E11'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-2', region=t.sets['bas'],
 u1=UNSET,u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E11'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-3', region=t.sets['gauche'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E11'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-4', region=t.sets['haut'],
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E22'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-5', region=t.sets['droite'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E22'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-6', region=t.sets['bas']   ,
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E22'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-7', region=t.sets['gauche'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E22'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-8', region=t.sets['haut'],
u1=UNSET, u2=l2, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E12'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-13', region=t.sets['droite'],
u1=UNSET, u2=l1/sqrt(2), ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E12'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-14', region=t.sets['bas'], u1=0.0,
u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E12'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-15', region=t.sets['gauche'],
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-E12'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-16', region=t.sets['haut'],
u1=l2/sqrt(2), u2=UNSET, ur3=UNSET)

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_1', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_1'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_1'].waitForCompletion()
```

Post-traitement des données après le calcul du premier ordre

```
area = mdb.models[modelname].rootAssembly.getArea(f)
odb = openOdb('Job_ordre_1.odb')
Charge = ['E11','E22','E12']
#intereting numbers
##Attention : the number of integration points and nodes are slightly different
noip = len(odb.steps['Step-E11'].frames[1].fieldOutputs['E'].values)
print('number of total integration points', noip)
non = len(odb.steps['Step-E11'].frames[1].fieldOutputs['U'].values)
print('number of total nodes', non)
noe = len(odb.steps['Step-E11'].frames[1].fieldOutputs['EVOL'].values)
print('number of total elements', noe)
#initialisation
E = numpy.zeros((noip,3,4))
S = numpy.zeros((noip,3,4))
U = numpy.zeros((non,3,2))
IVOL = numpy.zeros(noip)
EVOL = numpy.zeros(noe)
Coordinates = numpy.zeros((non,3)) ##t is the database instance (configurated in each model)
##extract data from the odb file
for i,charge in enumerate(Charge):
    Eodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['E' ].values
    Sodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['S' ].values
    Uodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['U' ].values
    for l in xrange(noip):
      E[l,i,:] = Eodb[l].data
      S[l,i,:] = Sodb[l].data
    for l in xrange(non):
        U[l,i,:] = Uodb[l].data
IVOLodb = odb.steps['Step-'+'E11'].frames[1].fieldOutputs['IVOL'].values
for i in xrange(noip):
  IVOL[i] = IVOLodb[i].data
EVOLodb = odb.steps['Step-'+'E11'].frames[1].fieldOutputs['EVOL'].values
for i in xrange(noe):
  EVOL[i] = EVOLodb[i].data
NodeArraySet=t.nodes
for i in xrange(non):
  Coordinates[i] = NodeArraySet[i].coordinates
```

Code de la matrice $[C^{hom}]$

```
Scalar = numpy.zeros((len(Charge),len(Charge)))
Stiffness = numpy.zeros((len(Charge),len(Charge)))
for i,charge in enumerate(Charge):
    Scalar[i,i] = 2*float(odb.steps['Step-'+charge].historyRegions['Assembly ASSEMBLY'].historyOutputs['ALLSE'].data[0][1])

```

```
Stiffness = numpy.array([[sum(S[:,0,0]*IVOL[:]), sum(S[:,1,0]*IVOL[:]), 0.0],
                [sum(S[:,0,1]*IVOL[:]), sum(S[:,1,1]*IVOL[:]), 0.0],
                [0.0            , 0.0            , sum(S[:,2,3]*IVOL[:]*sqrt(2))]])
Stiffness = Stiffness/(11*12*Amp)
Stiffness[numpy.abs(Stiffness)< 10**-8] = 0
print(' la matrice de raideur homogeneisee C hom',Stiffness_2)
```

Calcul de la précontrainte

```
# Calculate the initial deplacement by the reference to the pinned point
u_ini = numpy.zeros((non,2,3))
u_ini[:,0,0] = Coordinates[:,0] - referencepoint[0]
u_ini[:,1,1] = Coordinates[:,1] - referencepoint[1]
u_ini[:,0,2] = (Coordinates[:,1] - referencepoint[1])/sqrt(2)
u_ini[:,1,2] = (Coordinates[:,0] - referencepoint[0])/sqrt(2)

# Calculate u_1 by using the substraction
u_1 = numpy.transpose(U,(0,2,1)) - u_ini # Allouer directement avec les bons axes
elementslist = t.elements

# lanuch the interpolation for u_1 from nodes to integration points
u_1_inter = numpy.zeros((noip,2,3))
for i in xrange(noe):
    element_temp = elementslist[i]
    for l in range(4):
        # dummy average method
        adding=(u_1[element_temp.connectivity[0], :, :] + u_1[element_temp.connectivity[1], :, :] +
                u_1[element_temp.connectivity[2], :, :] + u_1[element_temp.connectivity[3], :, :] )
        u_1_inter[4*i+l, :, :] = adding/4

# Compute the average deplacement by adding up all areas associated with each integration point
u_1_aver = numpy.zeros((2,3))
for i in range(2):
    for j in range(3):
        u_1_aver[i][j] = sum(u_1_inter[:,i,j]*IVOL[:])
u_1_aver = u_1_aver/area

print('average displacement in ordre 1',u_1_aver)
# Calculate U_nabla_1
# Stop subtracting the average displacement according to the antisymmetry
U_nabla_1 = u_1_inter

# Construction of matrix U_nabla_1
Charge = ['K111','K221','K121','K112','K222','K122']
matrix_nabla = numpy.zeros((len(Charge),noip,3,6))

for i in range(len(Charge)):
    matrix_nabla[i,:,0,0:3] = U_nabla_1[:,0,0:3]
    matrix_nabla[i,:,2,0:3] = U_nabla_1[:,1,0:3]
    matrix_nabla[i,:,1,3:6] = U_nabla_1[:,1,0:3]
    matrix_nabla[i,:,2,3:6] = U_nabla_1[:,0,0:3]

# Construction of pre-deformation
Epsilon = numpy.zeros((len(Charge),noip,3))
for i in range(len(Charge)):
    Epsilon[i,:,:] = matrix_nabla[i,:,:,i]

# Sigma 11, Sigma 22, Sigma 12
Sigma = numpy.zeros((len(Charge),noip,4))
for i in range(len(Charge)):
    for j in xrange(noip):
        element_number = j/4
        Sigma[i,j,0:3] = numpy.dot(Matrix_C[element_number],Epsilon[i,j,:])
# Sigma 11, Sigma 22, Sigma33, Sigma 12
Sigma[:,:,3] = Sigma[:,:,2]
for i in range(len(Charge)):
    for j in xrange(noip):
        element_number = j/4
        Sigma[i,j,2] = ((Sigma[i,j,0]+Sigma[i,j,1])*
        (1-Matrix_S[element_number,0,0]*2/Matrix_S[element_number,2,2]))

#Storage integration points' stress to unf files
print('Storage integration points stress to unf files')

for i in range(4):
    for j in range(6):
        temp = FortranFile('prestress'+str(i+1)+str(j+1)+'.unf','w')
        for k in xrange(noip):
            temp.write_record(numpy.float32(Sigma[j][k][i]))
        temp.close()
```

Calcul des forces volumiques

```
import time
# from scipy.io import FortranFile
from MyFortranfile import FortranFile
#unf format storage
Charge = ['E11','E22','E12']

force11 = FortranFile('force11.unf', 'w')
force12 = FortranFile('force12.unf', 'w')
force13 = FortranFile('force13.unf', 'w')
force14 = FortranFile('force14.unf', 'w')
force15 = FortranFile('force15.unf', 'w')
force16 = FortranFile('force16.unf', 'w')
```

```
force21 = FortranFile('force21.unf', 'w')
force22 = FortranFile('force22.unf', 'w')
force23 = FortranFile('force23.unf', 'w')
force24 = FortranFile('force24.unf', 'w')
force25 = FortranFile('force25.unf', 'w')
force26 = FortranFile('force26.unf', 'w')

with file('S24.txt', 'w') as outfile_FORCE24:
# f plutot que s... correspond a la notation de Voigt [2x6]
    for k in xrange(noip):
    #STEP E11
        force11.write_record(numpy.float32(S[k,0,0]-(l1*l2/area)*Stiffness[0,0]))
        # numpy.savetxt(outfile_FORCE11,  array(S[k,0,0]-Stiffness[0,0]).reshape(1,), fmt='%-7.8E')
        force24.write_record(numpy.float32(S[k,0,1]-(l1*l2/area)*Stiffness[0,1]))
        force21.write_record(numpy.float32(S[k,0,3]))
        # numpy.savetxt(outfile_FORCE24,  array().reshape(1,), fmt='%-7.8E')
        force14.write_record(numpy.float32(S[k,0,3]))
    #STEP E22
        force12.write_record(numpy.float32(S[k,1,0]-(l1*l2/area)*Stiffness[0,1]))
        force25.write_record(numpy.float32(S[k,1,1]-(l1*l2/area)*Stiffness[1,1]))
        force22.write_record(numpy.float32(S[k,1,3]))
        force15.write_record(numpy.float32(S[k,1,3]))
    #STEP E12
        force13.write_record(numpy.float32(S[k,2,0]))
        force26.write_record(numpy.float32(S[k,2,1]))
        force23.write_record(numpy.float32(S[k,2,3]-(l1*l2/area)*Stiffness[2,2]/sqrt(2)))
        force16.write_record(numpy.float32(S[k,2,3]-(l1*l2/area)*Stiffness[2,2]/sqrt(2)))
force11.close()
force12.close()
force13.close()
force14.close()
force15.close()
force16.close()
force21.close()
force22.close()
force23.close()
force24.close()
force25.close()
force26.close()
```

Code de la matrice $[D]$

```
print('To calculate D')

D_list = numpy.zeros((noip,6,6))
for i in range(2):
  for j in range(3):
    for k in range(2):
      for l in range(3):
        D_list[:,i*3+j,k*3+l] = (U_nabla_1[:,0,j]*Matrix_C_new[:,2*i,2*k]*U_nabla_1[:,0,l] +
                    U_nabla_1[:,1,j]*Matrix_C_new[:,2*i+1,2*k]*U_nabla_1[:,0,l] +
                    U_nabla_1[:,0,j]*Matrix_C_new[:,2*i,2*k+1]*U_nabla_1[:,1,l] +
                    U_nabla_1[:,1,j]*Matrix_C_new[:,2*i+1,2*k+1]*U_nabla_1[:,1,l])
D = numpy.zeros((6,6))
for i in range(6):
  for j in range(6):
    D[i,j] = sum(D_list[:,i,j]*IVOL[:])/(l1*l2)
```

Problème du deuxième ordre et ses conditions aux limites

```
import time
m = mdb.models[modelname]
m.Stress(name = 'prestress',distributionType=USER_DEFINED, region=t.sets['Cellule'])
for ind,charge in enumerate(Charge):
    del m.steps['Step-'+charge]
mdb.models[modelname].PinnedBC(createStepName='Initial', localCsys=None,
    name='BC-25', region=mdb.models[modelname].rootAssembly.sets['Set-1'])

############## BCs ##################
#STEP-K111
m.StaticStep(name='Step-K111',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K111',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-1', region=t.sets['droite'],
u1= UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K111'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-2', region=t.sets['bas'],
 u1=UNSET,u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K111'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-3', region=t.sets['gauche'],
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K111'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-4', region=t.sets['haut'],
u1=UNSET, u2=0.0, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K111', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K111', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K111', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K1', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K1'].submit(consistencyChecking=OFF)
```

```
mdb.jobs['Job_ordre_2_K1'].waitForCompletion()
m.steps['Step-K111'].suppress()

#STEP-K221
m.StaticStep(name='Step-K221',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K221',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-5', region=t.sets['droite'],
u1= UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K221'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-6', region=t.sets['bas'],
u1=UNSET,u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K221'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-7', region=t.sets['gauche'],
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K221'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-8', region=t.sets['haut'],
u1=UNSET, u2=0.0, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K221', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K221', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K221', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K2', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K2'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_2_K2'].waitForCompletion()
m.steps['Step-K221'].suppress()

#STEP-K121
m.StaticStep(name='Step-K121',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K121',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-13', region=t.sets['droite'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K121'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-14', region=t.sets['bas'],
 u1=0.0,u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K121'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-15', region=t.sets['gauche'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K121'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-16', region=t.sets['haut'],
u1=0.0, u2=UNSET, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K121', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K121', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K121', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K3', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K3'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_2_K3'].waitForCompletion()
m.steps['Step-K121'].suppress()

#STEP-K112
m.StaticStep(name='Step-K112',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K112',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-17', region=t.sets['droite'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K112'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-18', region=t.sets['bas'],
 u1=0.0,u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K112'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-19', region=t.sets['gauche'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K112'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-20', region=t.sets['haut'],
u1=0.0, u2=UNSET, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K112', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K112', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K112', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K4', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K4'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_2_K4'].waitForCompletion()
m.steps['Step-K112'].suppress()

#STEP-K222
m.StaticStep(name='Step-K222',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K222',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-21', region=t.sets['droite'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K222'
```

```
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-22', region=t.sets['bas'],
u1=0.0,u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K222'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-23', region=t.sets['gauche'],
u1=0.0, u2=UNSET, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K222'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-24', region=t.sets['haut'],
u1=0.0, u2=UNSET, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K222', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K222', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K222', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K5', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K5'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_2_K5'].waitForCompletion()
m.steps['Step-K222'].suppress()

#STEP-K122
m.StaticStep(name='Step-K122',previous= 'Initial')
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K122',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-9', region=t.sets['droite'],
u1= UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K122'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-10', region=t.sets['bas'],
 u1=UNSET,u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K122'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-11', region=t.sets['gauche'],
u1=UNSET, u2=0.0, ur3=UNSET)
cc.DisplacementBC(amplitude=UNSET, createStepName='Step-K122'
    , distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None, name='BC-12', region=t.sets['haut'],
u1=UNSET, u2=0.0, ur3=UNSET)
m.FieldOutputRequest(createStepName='Step-K122', name='F-Output-2', variables=('S', 'E', 'U' , 'IVOL','EVOL','BF'))
m.HistoryOutputRequest(createStepName='Step-K122', name='H-Output-2', variables=PRESELECT)
m.BodyForce(comp1=1.0, comp2=1.0, comp3=1.0,
                    createStepName='Step-K122', distributionType=USER_DEFINED, field='', name=
                    charge+'BodyForce', region=t.sets['Cellule'])
mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=90, memoryUnits=PERCENTAGE, model=modelname, modelPrint=OFF,
    multiprocessingMode=DEFAULT, name='Job_ordre_2_K6', nodalOutputPrecision=SINGLE,
    numCpus=1, numGPUs=0, queue=None, scratch='', type=ANALYSIS,
    userSubroutine='DLOAD_SIGINI.for', waitHours=0, waitMinutes=0)
mdb.jobs['Job_ordre_2_K6'].submit(consistencyChecking=OFF)
mdb.jobs['Job_ordre_2_K6'].waitForCompletion()
m.steps['Step-K122'].suppress()
```

Post-traitement des données après le calcul du deuxième ordre

```
for i in range(6):
    openOdb('Job_ordre_2_K'+str(i+1)+'.odb')
Charge = ['K111','K221','K121','K112','K222','K122']

#initialisation
E_2 = numpy.zeros((noip,6,4))
S_2 = numpy.zeros((noip,6,4))
U_2 = numpy.zeros((non,2,6))
IVOL_2 = numpy.zeros(noip)
EVOL_2 = numpy.zeros(noe)
Coordinates_2 = numpy.zeros((non,3))

# Read data from six odb files
# number of integration points, elements, nodes remain constant in deux differents post-calculation
for i,charge in enumerate(Charge):
  odb = session.openOdb('Job_ordre_2_K'+str(i+1)+'.odb')
  Eodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['E' ].values
  Sodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['S' ].values
  Uodb = odb.steps['Step-'+charge].frames[1].fieldOutputs['U' ].values
  for l in xrange(noip):
    E_2[l,i,:] = Eodb[l].data
    S_2[l,i,:] = Sodb[l].data
  for l in xrange(non):
    U_2[l,:,i] = Uodb[l].data
odb = session.openOdb('Job_ordre_2_K1.odb')
IVOLodb = odb.steps['Step-'+'K111'].frames[1].fieldOutputs['IVOL'].values
for i in xrange(noip):
  IVOL_2[i] = IVOLodb[i].data
EVOLodb = odb.steps['Step-'+'K111'].frames[1].fieldOutputs['EVOL'].values
for i in xrange(noe):
  EVOL_2[i] = EVOLodb[i].data
NodeArraySet=t.nodes
for i in xrange(non):
  Coordinates_2[i] = NodeArraySet[i].coordinates
```

Calcul de la matrice $[F]$

```
print('To calculate F')
F = numpy.zeros((6,6))
for i in range(6):
  for j in range(6):
```

```
    temp = ( S_2[ :,i,0] * (S_2[ :,j,0]*Matrix_S[ :,0,0] + S_2[ :,j,1]*Matrix_S[ :,0,1])
        +S_2[ :,i,1] * (S_2[ :,j,0]*Matrix_S[ :,1,0] + S_2[ :,j,1]*Matrix_S[ :,1,1])
        +S_2[ :,i,3] * Matrix_S[ :,2,2] * S_2[ :,j,3] )
    F[i,j] = sum(temp*IVOL_2)/(11*12)
```

Calcul de la matrice $[G]$

```
print('To calculate G')
print('To calculate U_nabla_2')
#interpolation of U to U_inter at intergration points
elementslist = t.elements
U_inter = numpy.zeros((noip,2,6))
for i in xrange(noe):
    element_temp = elementslist[i]
    for l in range(4):
        if l == 0:
            a1 = -sqrt(3)/3
            a2 = -sqrt(3)/3
        elif l == 1:
            a1 =  sqrt(3)/3
            a2 = -sqrt(3)/3
        elif l == 2:
            a1 = -sqrt(3)/3
            a2 =  sqrt(3)/3
        elif l == 3:
            a1 =  sqrt(3)/3
            a2 =  sqrt(3)/3
        U_inter[4*i+l, :, :] =  (
            U_2[element_temp.connectivity[0], :, :]*(1.0/4)*(1.0-a1)*(1.0-a2) +
            U_2[element_temp.connectivity[1], :, :]*(1.0/4)*(1.0+a1)*(1.0-a2) +
            U_2[element_temp.connectivity[2], :, :]*(1.0/4)*(1.0+a1)*(1.0+a2) +
            U_2[element_temp.connectivity[3], :, :]*(1.0/4)*(1.0-a1)*(1.0+a2))

U_inter_aver = numpy.zeros((2,6))
for i in range(2):
  for j in range(6):
    U_inter_aver[i,j] = sum(U_inter[ :,i,j]*IVOL_2[ :])/area
# antisymmetry
U_inter_aver[1,0] = 0.0
U_inter_aver[1,1] = 0.0
U_inter_aver[0,2] = 0.0
U_inter_aver[0,3] = 0.0
U_inter_aver[0,4] = 0.0
U_inter_aver[1,5] = 0.0

print(U_inter_aver)

U_nabla_2 = numpy.zeros((noip,2,6))
for i in range(2):
  for j in range(6):
    U_nabla_2[ :,i,j] = U_inter[ :,i,j] - U_inter_aver[i,j]

# read stress from the results in the calculation ordre 1
B = numpy.zeros((noip,2,6))
temp = numpy.transpose(S,(0,2,1))
B[ :,0,0 :3] = temp[ :,0,0 :3]
B[ :,0,3 :6] = temp[ :,3,0 :3]
B[ :,1,0 :3] = temp[ :,3,0 :3]
B[ :,1,3 :6] = temp[ :,1,0 :3]

#calculate G
G = numpy.zeros((6,6))
for i in range(6):
  for j in range(6):
    temp = numpy.zeros(noip)
    for l in range(2):
      temp = temp + B[ :,l,i]*U_nabla_2[ :,l,j]
    G[i,j] = sum(temp*IVOL_2)/(11*12)
```

Calcul de la matrice $[H]$

```
print('To calculate H')
H_list = numpy.zeros((noip,12,12))

for i in range(2):
  for j in range(6):
    for k in range(2):
      for l in range(6):
        H_list[ :,i*6+j,k*6+l] = (U_nabla_2[ :,0,j]*Matrix_C_new[ :,2*i,2*k]*U_nabla_2[ :,0,l] +
                    U_nabla_2[ :,1,j]*Matrix_C_new[ :,2*i+1,2*k]*U_nabla_2[ :,0,l] +
                    U_nabla_2[ :,0,j]*Matrix_C_new[ :,2*i,2*k+1]*U_nabla_2[ :,1,l] +
                    U_nabla_2[ :,1,j]*Matrix_C_new[ :,2*i+1,2*k+1]*U_nabla_2[ :,1,l])
H = numpy.zeros((12,12))
for i in range(12):
  for j in range(12):
    H[i,j] = sum(H_list[ :,i,j]*IVOL_2[ :])/(11*12)
```

La symétrie et l'anti-symétrie de $[D]$, $[F]$, $[G]$ et $[H]$

```
numpy.savetxt(str(r)+'_C.csv',Stiffness,delimiter=';')
D = symmetry_null_6_6(D)
print('D',D,)
numpy.savetxt(str(r)+'_D.csv',D,delimiter=';')
F = symmetry_null_6_6(F)
```

```
print('F',F)
numpy.savetxt(str(r)+'_F.csv',F,delimiter=';')
G = symmetry_null_6_6(G)
print('G',G)
numpy.savetxt(str(r)+'_G.csv',G,delimiter=';')
J = F - G.transpose() - G
print('J',J)
numpy.savetxt(str(r)+'_J.csv',J,delimiter=';')
LL = (J[4,4]/Stiffness[1,1])**(0.5)
results_list.append(LL)

os.remove('Job_ordre_1.lck')
for i in range(6):
  os.remove('Job_ordre_2_K'+str(i+1)+'.lck')
openOdb('Job_ordre_1.odb').close()
for i in range(6):
  openOdb('Job_ordre_2_K'+str(i+1)+'.odb').close()
```

```
def symmetry_null_6_6(matrix):
  matrix[2:5,0:2] = 0
  matrix[0:2,2:5] = 0
  matrix[2:5,5] = 0
  matrix[5,2:5] = 0

  return matrix
def symmetry_null_12_12(matrix):
  # Bloc 11
  matrix[2:5,0:2] = 0
  matrix[0:2,2:5] = 0
  matrix[2:5,5] = 0
  matrix[5,2:5] = 0
  # Bloc 12
  matrix[0:2,6:8] = 0
  matrix[2:5,8:11] = 0
  matrix[5,0:8] = 0
  matrix[0:2,11] = 0
  matrix[5,11] = 0
  # Bloc 21
  matrix[6:8,0:2] = 0
  matrix[8:11,2:5] = 0
  matrix[11,0:2] = 0
  matrix[6:8,5] = 0
  matrix[11,5] = 0
  # Bloc 22
  matrix[8:11,6:8] = 0
  matrix[6:8,8:11] = 0
  matrix[8:11,11] = 0
  matrix[11,8:11] = 0

  return matrix
```

Fortran 77 (Subroutines implémentées dans Abaqus)

```
C
C
C
      SUBROUTINE DLOAD(F,KSTEP,KINC,TIME,NOEL,NPT,LAYER,KSPT,
     1 COORDS,JLTYP,SNAME)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION TIME(2), COORDS(3)
      CHARACTER*80 SNAME, stepnumber*10, direction*10
      CHARACTER*256 OUTDIR
      CHARACTER*256 JOBNAME
C       INTEGER*4 getcwd, status
C       character*255 dirname
       real ::R
       integer ::IP, flags
       CALL GETOUTDIR( OUTDIR, LENOUTDIR )
       CALL GETJOBNAME( JOBNAME, LENJOBNAME )
       IP = 4*(NOEL-1)+NPT
       flag= 100+ (7-KSTEP)*3+JLTYP
       write(stepnumber, '(i1)') KSTEP
       write(direction, '(i1)') JLTYP
       if (IP.eq.1) then
      OPEN(unit=flag,file=trim(OUTDIR)//'\force'//trim(direction)//
     1JOBNAME(14:14)//'.unf', form="unformatted", acc
     2ess="stream")
       else
          continue
       endif
       read(flag, POS=(IP*5 + (IP-1)*7)) R
       F = R
C       write(7,*) NOEL,NPT,JLTYP,KSTEP,F,(IP*5 + (IP-1)*7),'s'//
C     1trim(direction)//trim(stepnumber)
       RETURN
       END
C
      SUBROUTINE SIGINI(SIGMA,COORDS,NTENS,NCRDS,NOEL,NPT,LAYER,
     1 KSPT,LREBAR,NAMES)
C
```

```
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION SIGMA(NTENS),COORDS(NCRDS)
      CHARACTER NAMES(2)*80
      CHARACTER*256 OUTDIR
      CHARACTER*256 JOBNAME

       real :: R1, R2, R3, R4
      CALL GETOUTDIR( OUTDIR, LENOUTDIR )
      CALL GETJOBNAME( JOBNAME, LENJOBNAME )
       IP = 4*(NOEL-1)+NPT
       flag= 100+ (7-KSTEP)*4
       if (IP.eq.1) then
      OPEN(unit=flag,file=trim(OUTDIR)//'\prestress1'//JOBNAME(14:14)
     1//'.unf', form="unformatted", access="stream")
      OPEN(unit=flag+1,file=trim(OUTDIR)//'\prestress2'//JOBNAME(14:14)
     1//'.unf', form="unformatted", access="stream")
      OPEN(unit=flag+2,file=trim(OUTDIR)//'\prestress3'//JOBNAME(14:14)
     1//'.unf', form="unformatted", access="stream")
      OPEN(unit=flag+3,file=trim(OUTDIR)//'\prestress4'//JOBNAME(14:14)
     1//'.unf', form="unformatted", access="stream")
       else
           continue
       endif
       read(flag,    POS=(IP*5 + (IP-1)*7)) R1
       read(flag+1, POS=(IP*5 + (IP-1)*7)) R2
       read(flag+2, POS=(IP*5 + (IP-1)*7)) R3
       read(flag+3, POS=(IP*5 + (IP-1)*7)) R4
      SIGMA(1) = R1
      SIGMA(2) = R2
      SIGMA(3) = R3
      SIGMA(4) = R4
       write(7,*) NTENS,NOEL,NPT,COORDS,SIGMA(1),SIGMA(2),SIGMA(4)
      RETURN
      END
```